Cloudera Stream Processing Community Edition 2.0.0

Getting Started Guide

Date published: 2022-07-29
Date modified: 2023-01-31

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Getting started with Streams Messaging 4

Getting started with Kafka 4
- Creating a topic 4
- Producing and consuming data 5
- Monitoring Kafka topics with SMM 6

Getting started with Schema Registry 9
- Creating a schema 9
- Evolving a schema 11
- Deleting a schema 13
- Integrating Schema Registry with a Kafka Java client 13

Getting started with Kafka Connect 16
- Creating a database and table in PostgreSQL 17
- Creating a topic and producing messages 17
- Deploying the connector 18
- Monitoring and managing the connector 21

Getting Started with Streams Messaging Manager Alert Policies 23

Setting up connectors that use JDBC drivers 26

Getting started with Streaming Analytics 29

Getting started with SQL Stream Builder 29
- 1. Creating a project 29
- 2. Creating a job and table 31
- 3. Running a SELECT query 35
- 4. Creating and querying a view 36
- 5. Outputting results of SQL query 37

Using different connectors in SSB 51
- Using Filesystem source 51
- Using PostgreSQL CDC connector 53

Monitoring SQL results and jobs 54
- Sampling results in Streaming SQL Console 54
- Reviewing data in Streams Messaging Manager 54
- Reviewing jobs in Flink Dashboard 55

Managing SQL jobs 57
- Configuring SQL job settings 57
- Managing SQL jobs 59
- Loading SQL queries to SQL Editor 60
- Using SQL Stream Builder REST API 62
- Materialized View Pagination 62

Getting started with Streams Messaging

Get started with Streams Messaging in CSP Community Edition.

The following collection of tutorials walks you through the basic usage and features of Kafka, Schema Registry, Streams Messaging Manager, and Kafka Connect. Complete any of the following tutorials to get familiar with the features and functionalities of the Streams Messaging components.

Getting started with Kafka

Get started with Kafka in CSP Community Edition.

The following sections walk you through the basics of how Kafka is used in CSP Community Edition. Completing this tutorial you will learn:

- How to create a topic using Streams Messaging Manager (SMM).
- How to produce and consume data using Kafka's built-in console clients as well as your own client applications.
- How to monitor topic activity in SMM.

Before you begin

Find out the name or ID of the Kafka container. You will need to pass the container name or ID as a parameter in some of the commands you will be running.The container name and ID can be listed using docker ps. For example:

```
docker ps -a --format '{{.ID}}\t{{.Names}}' --filter "name=kafka.(\d)"
```

The Kafka container will either be called cspce-kafka-1 or cspce_kafka_1.

Creating a topic

Learn how to create a Kafka topic using the SMM UI.

Procedure

1. Access the SMM UI by entering the following in a browser window:

   ```
   http://localhost:9991
   ```

2. Click (Topics) in the navigation sidebar.
3. Click Add New.
4. Configure the topic as follows:
   - Topic Name: csp-ce
   - Partitions: 3
   - Availability: Low
   - Cleanup Policy: delete

   Important: You can only create topics using the Low preset or the Custom preset with REPLICATION FACTOR and MIN.INSYNC.REPLICAS set to 1. This is because CSP Community Edition has a single Kafka broker deployed. Additionally, the name of the topic can only contain ASCII alphanumeric characters (a-z, A-Z, 0-9), dots (.), underscores(_), and minus signs (-).
5. Click Save.

6. Verify that the topic was created.

   This can be done by typing csp-ce in the search field. If the topic was successfully created, it will be listed under Topics.

Producing and consuming data

Learn how to produce and consume data to and from Kafka topics using Kafka's built-in console tools or your own client application.

About this task

Once you have created your topic it's time to start producing (writing) and consuming (reading) some data (also referred to as records or messages). Data production and consumption happens using Kafka producer and consumer applications, or clients for short. Clients connect to a Kafka server, called a broker, and either produce data to or consume data from topics. In most production environments these applications are custom built using a Kafka client library. However, Kafka is shipped with command line tools, including a console producer and consumer application that you can use to test Kafka's features and capabilities.

The following list of steps will walk you through how you can use Kafka's built-in console producer to produce some messages and then consume and view those messages using the SMM UI. Additionally, information about how to configure your own custom developed clients to connect to Kafka in CSP Community Edition is also provided.

Before you begin

- Ensure that you have a topic available. If not, create one, see Creating a topic on page 4.
- Ensure that you have the SMM UI open. If not, enter the following in a browser window:

```
http://localhost:9991
```

Steps

For Console Client

1. Open a terminal sessions and run the following command:

   ```
   docker exec -it [***KAFKA CONTAINER NAME OR ID***] /bin/bash
   ```

   This command launches a Bash session within the docker container that Kafka is running in. The interactive session is required to run the console producer.
2. Run the console producer to start producing data.

   ```
   /opt/kafka/bin/kafka-console-producer.sh --bootstrap-server localhost:90
   94 --topic csp-ce
   ```

   The --bootstrap-server option specifies the host and port of the Kafka broker that the client should connect to. The --topic option specifies the topic that data will be produced to.
3. Start typing to produce data.

   ```
   >my first message
   >my second message
   ```

   Once you are done entering messages, ensure that you leave the console producer running.
4. In the SMM UI locate your topic and click (Profile) next to the topic name.

5. Go to Data Explorer.

   The Data Explorer tab lets you sample the data that is flowing through the topic. The messages that you produced using the console producer will be available on this page. Select different partitions and experiment with the slider controls to view data.

   Note: The Data Explorer defaults to Partition 0 of a topic. If you do not see any data, select a different partition from the Partitions drop-down to find the messages that you produced.

6. Switch over to the terminal session running the producer and produce some more data.
7. Switch back to the Data Explorer in SMM and refresh the page.

   The new messages you produced should be visible in Data Explorer.
8. Close the producer session with CTRL+C.

For Custom Client

To connect your own Kafka clients, that are running outside of the docker network, to the Kafka broker deployed with CSP Community Edition you need to run the client with the following configuration:

```
bootstrap.servers=localhost:9094
```

Once you have a producer set up and running, you can immediately view the data that is being produced into a topic using the SMM UI.

1. In the SMM UI locate your topic and click (Profile) next to the topic name.
2. Go to Data Explorer.

   The Data Explorer tab lets you sample the data that is flowing through the topic. The messages that you produce will be available on this page. Select different partitions and experiment with the slider controls to view data.

   Note: The Data Explorer defaults to Partition 0 of a topic. If you do not see any data, select a different partition from the Partitions drop-down to find the messages that you produced.

Monitoring Kafka topics with SMM

Learn how you can monitor Kafka topics and topic activity in SMM.

About this task

Now that you know how to create a topic and produce/consume data, it is time to learn how you can monitor your topics and the activity of those topics using Streams Messaging Manager (SMM).

Although you already have a topic with some data, the following steps will walk you through another production/consumption process where data production and consumption is continuous. This is done so that the topic activity is closer to what you would actually see in a production environment. Once you have the data flowing, you will access the SMM UI and monitor topic and client activity as well as look at the data that is being produced.

Note: SMM can be used to monitor the activity of all types of Kafka entities, not just topics. For a comprehensive overview and more information regarding SMM, see the *Related Information* section at the bottom of this page.

Before you begin

- Ensure that you have a topic available. If not, create one, see Creating a topic on page 4.
- Ensure that you have the SMM UI open. If not, enter the following in a browser window:

  ```
  http://localhost:9991
  ```

- Ensure that you have two interactive Bash sessions open in the Kafka container. If not, open two separate terminal sessions and run the following command in both sessions:

```
docker exec -it [***KAFKA CONTAINER NAME OR ID***] /bin/bash
```

Procedure

1. Produce and consume some data.
   a) Pick a terminal session and run the console producer.

   ```
   vmstat 1 1000 | /opt/kafka/bin/kafka-console-producer.sh --bootstrap-ser
   ver localhost:9094 --topic csp-ce
   ```

   This command uses vmstat to generate data. The data that vmstat prints is then picked up by the console producer and is streamed into the specified Kafka topic. Each line from the output of vmstat is produced as a standalone message. Do not close this session or interrupt the producer, otherwise data production will stop.

   b) Switch over to the other terminal session and run the console consumer.

   ```
   /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:90
   94 --topic csp-ce --from-beginning --group my-consumer-group
   ```

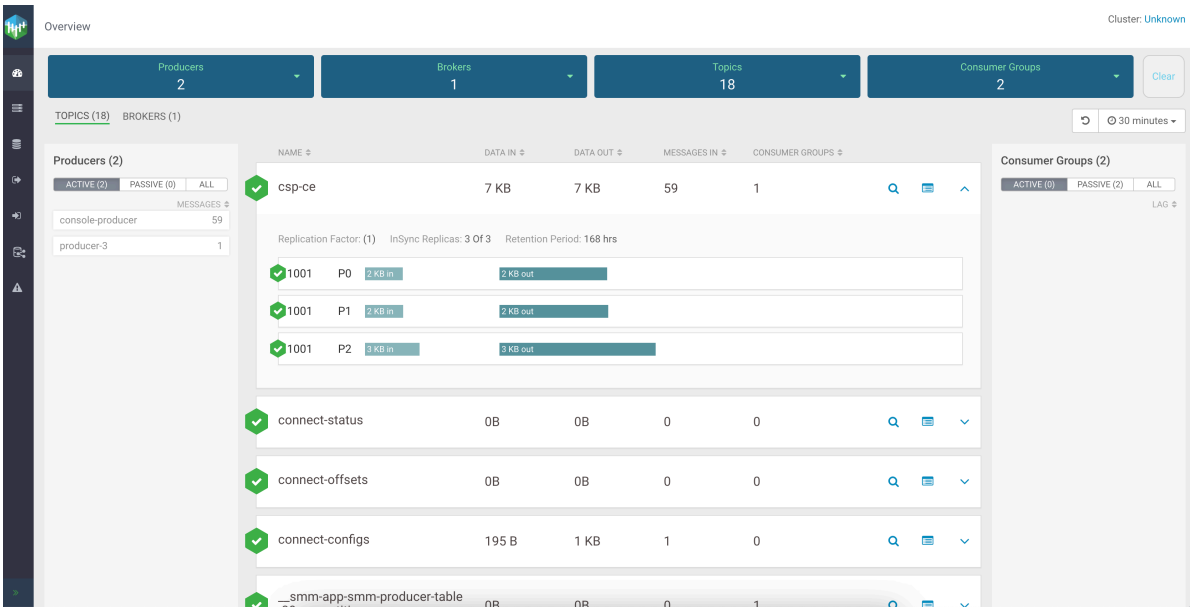
   The messages that are being produced by the console producer instance should appear on the screen. As long as you keep both processes running, and as long as vmstat is generating data, the producer will continue to produce messages, and the consumer will continue to consume them.

2. Switch over to the SMM UI and start monitoring topic activity.

   When you open SMM, you are presented with the Overview page. This page gives you information about the total number of producers, brokers, topics, and consumer groups. It also provides more detailed metrics about producers and consumers. The following points give a quick introduction of the UI and its features. Feel free to experiment and explore.

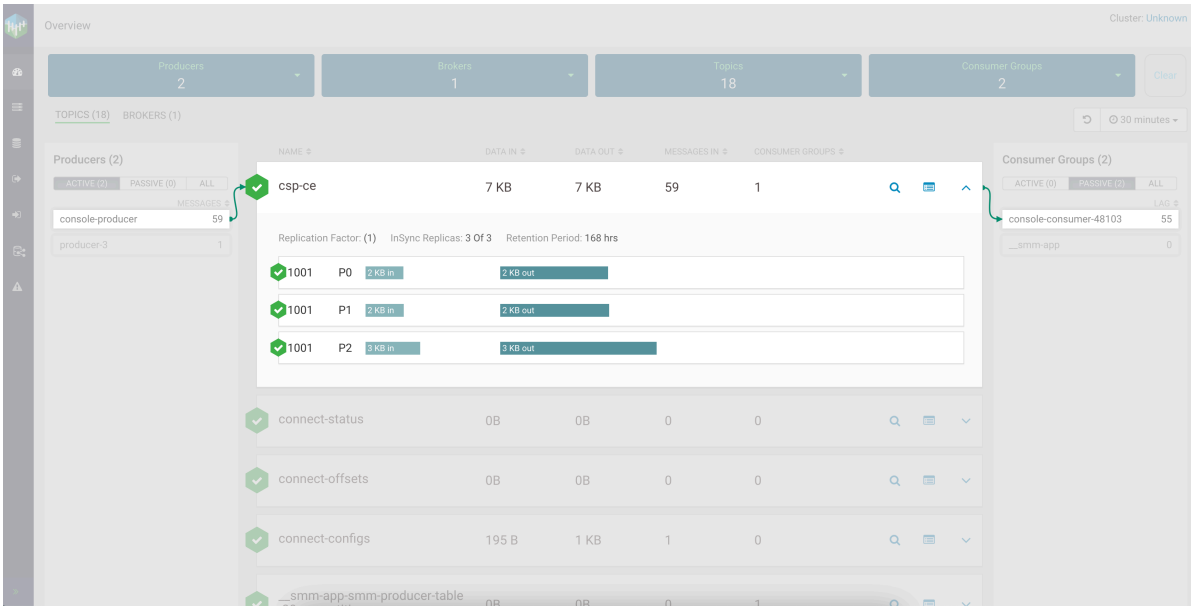
   Note: It may take a few minutes for recent activity to start appearing. If you do not see any activity, wait a few minutes and refresh the page.

   - In the middle you have a list of topics. Here, you can identify and review some key metrics for the overall topic activity. Clicking on a topic lists the partitions of that topic as well as partition-level metrics.



   - On the left and right hand side of the page you can see your producers and consumers. If everything is working correctly, you should see at least one active producer and at least one active consumer. Clicking on either the

producer or consumer will show their activity, highlighting all the topics and partitions they are writing to or reading from, respectively.



- Clicking on (Profile) redirects you to the topic details page which has a total of four tabs. The tabs and the information they present are as follows:

For Metrics

The Metrics tab collects all the topic-related metrics and utilization charts.

Tip: You can delete a topic on this tab by clicking ActionsDelete.

For Data Explorer

The Data Explorer tab lets you sample the data that is flowing through the topic. Select different partitions and experiment with the slider controls to view data. If the topic you are viewing is getting data produced into it actively, you can refresh the page to see the latest messages being produced

Tip: The Data Explorer tab of a topic can be accessed from the Overview page by clicking next to the name of a topic.

For Configs

The Configs tab gives you an overview of the topic's configuration properties. Additionally, you can also reconfigure a topic on this tab.

For Latency

The Latency tab gives you a powerful snapshot of end-to-end latency including the number of consumer groups for a topic, number of clients inside a particular consumer group, and number of partitions in a topic.This tab also includes two graphs that visualize messages consumed as well as end-to-end latency. By default the Latency tab will not show any data. This is because in order for SMM to fetch metrics related to end-to-end latency, clients must be configured. Specifically, interceptors must be enabled within the client. This is not covered in this tutorial. For more information on enabling interceptors and monitoring end-to-end latency, see *Monitoring End-to-End Latency using Streams Messaging Manager*.

What to do next

Completing this tutorial you learned how to create Kafka topics using SMM, how to produce and consume data and got familiar with the basic monitoring features of SMM. Cloudera recommends that you hop over to the Schema Registry or Kafka Connect tutorials and complete those as well. Alternatively, If you want to dive deeper into Kafka or SMM, visit any of the resources listed in the Related Information section below.

Related Information

Official Apache Kafka Documentation
Cloudera Runtime - Apache Kafka Overview
Cloudera Runtime - Developing Apache Kafka Applications
Cloudera Runtime - Monitoring Kafka Clusters using Streams Messaging Manager
Cloudera Runtime - Managing Kafka Topics using Streams Messaging Manager
Cloudera Runtime - Monitoring End-to-End Latency using Streams Messaging Manager
Streams Messaging Manager Alert Policies

Getting started with Schema Registry

Get started with Schema Registry in CSP Community Edition.

Schema Registry provides a shared repository of schemas that allows applications to flexibly interact with each other. Applications frequently need a way to share metadata across data format, schema, and semantics. Schema Registry addresses these challenges by evolving schemas such that a producer and consumer can understand different versions of the schemas but still read all information shared between both versions and safely ignore the rest.

The following sections teach you how you can:

- Create a schema.
- Evolve a schema.
- Delete a schema.
- Integrate your own Java Kafka client with Schema Registry.

Creating a schema

Learn how to create a schema in Schema Registry using the Schema Registry UI.

About this task

Before you begin

Creating a schema involves configuring the properties of the schema and uploading the schema itself to Schema Registry. Cloudera recommends that you use your own schema if one is available to you. If you do not have a schema available, an example schema is provided in Step 4 on page 10.

Procedure

1. Access the Schema Registry UI by entering the following in a browser window:

   ```
   http://localhost:7788/
   ```

2. Click +.

   The **Add New Schema** dialog appears.

3. Configure the schema.

   If you do not have a schema available and plan to use the example schema provided in the next step, provide a NAME and DESCRIPTION for the schema and leave all other properties set to their default values.

   NAME

   Add a unique name for the schema. Used as a key to look up schemas.

   DESCRIPTION

   Add a short description of the schema

   SCHEMA TYPE

   The schema format. Supported formats are Avro and JSON. If you do not have a schema available and plan to use the example provided in this guide, select AVRO

   SCHEMA GROUP

   Add the name of a schema group. Schema Groups allow you to group schemas in any logical order.

   COMPATIBILITY

   Sets the compatibility policy for the schema. A compatibility policy defines the rules of how a schema can be evolved. Once set, this cannot be changed. The policies are as follows:

   - FORWARD - allows evolving a schema by adding information.
   - BACKWARD - allows evolving a schema by removing information.

     Note: It is possible to add new fields to BACKWARD compatible schemas. In a case like this however, a default value must be provided for that field. An example of this is shown in the *Evolving a Schema* section.
   - BOTH - allows evolving a schema by adding and removing information.
   - NONE - no policy is applied to the schema, any change to the schema is accepted, compatibility checks are not carried out.

   For a comprehensive description on each policy, see *Compatibility Policies*.

   EVOLVE

   Controls whether a schema can be evolved later by creating multiple versions. If you deselect this option the schema can only have one version. Leave this set as the next section walks you through schema evolution using the schema you are creating now.

4. Copy and paste, drag and drop, or browse for your schema in the **SCHEMA TEXT** area.

   You can use the following example schema if you don't have a schema available.

   Tip: Although the following example is written in JSON, it is an Avro schema. This is because Avro schemas are created using the JSON format.

```
{
  "type": "record",
  "namespace": "com.cloudera.examples",
  "name": "MachineData",
  "doc": "Machine utilization metrics",
  "fields": [
    {
      "name": "timestamp",
      "doc": "Metrics timestamp",
      "type": "long"
    },
    {
      "name": "hostname",
      "doc": "Host name of the source machine",
      "type": "string"
    },
    {
      "name": "counters",
```

```
      "doc": "Machine counters",
      "type": {
        "type": "array",
        "items": {
          "type" : "record",
          "name" : "Counter",
          "fields" : [
            {
              "name": "name",
              "doc": "Name of the counter",
              "type": "string"
            },
            {
              "name": "value",
              "doc": "Value of the counter",
              "type": "long"
            },
            {
              "name": "unit",
              "doc": "Unit of the counter value",
              "type": ["string", "null"],
              "default": "null"
            }
          ]
        }
      }
    }
  ]
}
```

5. Click Save.

Evolving a schema

Learn how to evolve a schema in Schema Registry using the Schema Registry UI.

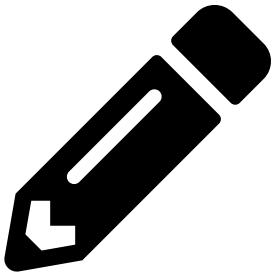
About this task

You evolve a schema when you create a new version of it. Schema Registry tracks the changes made to your schema, and stores each set of changes in a separate version of the schema. When multiple versions exist, you can select which version you want to use. Ensure that you understand compatibility policies, as they determine how you can evolve your schemas.

Procedure

1. Access the Schema Registry UI by entering the following in a browser window:

```
http://localhost:7788/
```

2. Identify and select the schema that you want to evolve (version).
3. Click the  icon to open the **Edit Version** dialog

4. Add a description of what has changed in this new version of the schema.

   You can view the description in the Schema Registry UI to easily understand what has changed in each version of the schema. As a result, Cloudera recommends that you add as much detail as you can.

5. Edit the schema in either of the following ways:

   - Edit it directly in the **SCHEMA TEXT** area.
   - Click Clear and upload a new version.

   If you used the MachineData schema provided as an example in *Creating a Schema*, add the following schema. This is the evolved version of the Machine data schema.

```
{
  "type": "record",
  "namespace": "com.cloudera.examples",
  "name": "MachineData",
  "doc": "Machine utilization metrics",
  "fields": [
    {
      "name": "timestamp",
      "doc": "Metrics timestamp",
      "type": "long"
    },
    {
      "name": "hostname",
      "doc": "Host name of the source machine",
      "type": "string"
    },
    {
      "name": "os_type",
      "doc": "OS type of the source machine",
      "type": "string",
      "default": "UNKNOWN"
    },
    {
      "name": "counters",
      "doc": "Machine counters",
      "type": {
        "type": "array",
        "items": {
          "type" : "record",
          "name" : "Counter",
          "fields" : [
            {
              "name": "name",
              "doc": "Name of the counter",
              "type": "string"
            },
            {
              "name": "value",
              "doc": "Value of the counter",
              "type": "long"
            },
            {
              "name": "unit",
              "doc": "Unit of the counter value",
              "type": ["string", "null"],
              "default": "null"
            }
          ]
        }
      }
    }
```

```
  ]
}
```

6. Click Save.
7. The schema you created will be visible on the UI.
8. Click COMPARE VERSIONS.

   In the **COMPARE VERSIONS** dialog you can compare versions of the schemas. If you have used the MachineData schema example, you will see that in MachineData V2 a new field regarding OS type was added.

```
"name": "os_type",
"doc": "OS type of the source machine",
"type": "string",
"default": "UNKNOWN"
```

   Notice that a default value of UNKNOWN is added. This is only needed if the schema compatibility is set to BACKWARD (default). New fields can only be added to backward compatible schemas if a default value is also provided for that field.

Deleting a schema

Learn how to delete a schema from Schema Registry with the Swagger UI or curl.

About this task

Schemas can be deleted if you no longer need them. However, unlike schema creation and evolution, schema deletion can not be done using the Schema Registry UI. Instead, you must use the Schema Registry API. The following provides instructions on how to delete a schema using the Swagger UI or curl.

Steps

For Swagger UI

1. Go to http://localhost:7788/swagger
2. Click Schema.
3. Click DELETE /api/v1/schemaregistry/schemas/{name}.
4. Click Try it out.
5. Enter the name of the Schema that you want to delete in the Name field.
6. Click Execute.

For curl

1. Open a new terminal session.
2. Run the following command:

```
curl -X DELETE "http://localhost:7788/api/v1/schemaregistry/sche
mas/[***SCHEMA NAME***]" -H "accept: application/json"
```

   Replace *[***SCHEMA NAME***]* with the name of the schema that you want to delete

Integrating Schema Registry with a Kafka Java client

Learn how you can integrate existing Kafka Java clients with Schema Registry.

About this task

The following steps walk you through how you can integrate an existing Java Kafka client with Schema Registry. Additionally, you will learn about the `KafkaAvroSerializer` and `KafkaAvroDeserializer`. These are the

Cloudera recommended converters that you should use when integrating clients with Schema Registry. Lastly, you will also learn how you can view the Avro data produced into a topic using Streams Messaging Manager (SMM)

Procedure

1. Set up dependencies.

   Any client application that you want to integrate with Schema Registry requires the following dependencies.

   a) Add the Cloudera Runtime Maven repository to the pom.xml of your project.

   ```
   <repositories>
           <repository>
               <id>Cloudera Libs</id>
               <url>https://repository.cloudera.com/artifactory/libs-rel
   ease-local/</url>
           </repository>
       </repositories>
   ```

   b) Add a schema-registry-serdes dependency to the pom.xml of your project.

   ```
   <dependency>
         <groupId>com.hortonworks.registries</groupId>
         <artifactId>schema-registry-serdes</artifactId>
   </dependency>
   ```

   c) Import the following classes.

   ```
   import com.hortonworks.registries.schemaregistry.SchemaVersionInfo;
   import com.hortonworks.registries.schemaregistry.client.SchemaRegistryCl
   ient;
   import com.hortonworks.registries.schemaregistry.errors.SchemaNotFound
   Exception;
   import static com.hortonworks.registries.schemaregistry.serdes.avro.A
   vroSnapshotSerializer.SERDES_PROTOCOL_VERSION;
   import static com.hortonworks.registries.schemaregistry.serdes.avro.S
   erDesProtocolHandlerRegistry.CURRENT_PROTOCOL
   ```

2. Configure producers and consumers.

   Producers and consumers require the following configuration properties to be set respectively.

   For Producer

   ```
   schema.registry.url=http://localhost:7788
   key.serializer=org.apache.kafka.common.serialization.StringSerializer
   value.serializer=com.hortonworks.registries.schemaregistry.serdes.avro.k
   afka.KafkaAvroSerializer
   ```

   For Consumer

   ```
   schema.registry.url=http://localhost:7788
   key.deserializer=org.apache.kafka.common.serialization.StringDeseriali
   zer
   ```

```
value.deserializer=com.hortonworks.registries.schemaregistry.serdes.avro
.kafka.KafkaAvroDeserializer
```

If you are specifying your configuration properties directly within the client application, your code will look similar to the following examples after configuration is complete.

For Producer

```
config.putAll(Collections.singletonMap(SchemaRegistryClient.Configuratio
n.SCHEMA_REGISTRY_URL.name(), props.get(SCHEMA_REGISTRY_URL)));
config.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.
class.getName());
config.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, KafkaAvroSeri
alizer.class.getName());
```

For Consumer

```
config.putAll(Collections.singletonMap(SchemaRegistryClient.Configuratio
n.SCHEMA_REGISTRY_URL.name(), props.get(SCHEMA_REGISTRY_URL)));
config.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeseriali
zer.class.getName());
config.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, KafkaAvroDe
serializer.class.getName());
```

The schema.registry.url property specifies where the Schema Registry server is running. In CSP Community Edition this is fixed and will always be http://localhost:7788.

The serializer/deserializer properties specify what classes the client should use to serialize or deserialize (convert) data. In this specific example key conversion is done using the default `StringSerializer` and `StringDeserializer` shipped with Kafka. Value conversion on the other hand is done using the `KafkaAvroSerializer` and `KafkaAvroDeserializer` respectively. Both of these are Cloudera provided and are Avro specific. They make it easy to write and read Avro objects to or from Kafka as serialization and deserialization are handled transparently. As a result, they are the Cloudera recommended converters that you should use when you are working with Avro data and are integrating with Schema Registry.

Both classes are Schema Registry aware. When the `KafkaAvroSerializer` is used to produce data to a Kafka topic, the serializer will first check with the configured Schema Registry service to ensure the schema being used is compatible with the schema registered for that topic. More specifically one of the following happens:

- If the topic has no schema registered, the serializer will register the schema on behalf of the client.
- If there’s already a schema registered for the topic, the serializer will check if the schema being used by the client is either the same or a compatible evolution of that schema.
  - If the schema is the same, the producer will use it for producing messages.
  - If the schema is a new version of the currently registered schema that is compatible with the previous versions, it will be registered with a new version number and will be used for producing messages.
- If the schema being used by the client is not compatible with the currently registered schema the serializer will throw an exception.

Additionally, the serializer also adds the schema's identifier from Schema Registry to the message payload, so that the consumer will know what's the correct schema version to fetch.

Afterwards, when the consumer is consuming the messages with the `KafkaAvroDeserializer`, the deserializer will fetch that schema using the identifier from the message and will use the schema to correctly deserialize the payload.

3. Build and run your clients to start producing and consuming data.

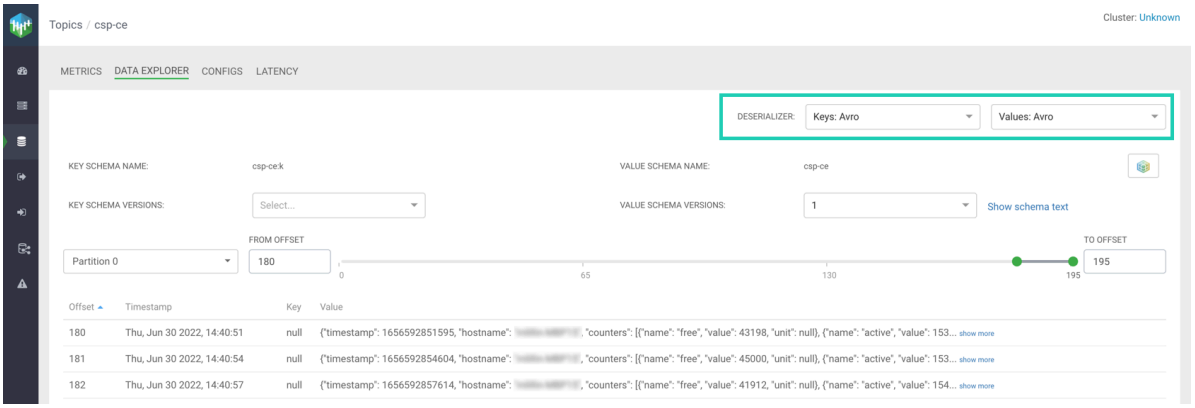
4. View data in SMM.
   a) Access the SMM UI by entering the following in a browser window:

```
http://localhost:9991
```

   b) Click (Topics) in the navigation sidebar.
   c) Find the topic you have been using and click next to the name of a topic.
   d) Go to Data Explorer.

   The messages the producer is producing into the topic should be visible on this page. However, the contents of the messages will not be readable.

   This is because your producer is producing binary data. However because SMM integrates with Schema Registry, it is capable of decoding the binary data using the schema registered for the topic in Schema Registry. This is done by selecting Avro for the key and/or value deserializer.



Getting started with Kafka Connect

Get started with Kafka Connect in CSP Community Edition.

Kafka Connect is a tool for streaming data between Apache Kafka and other systems in a reliable and scalable fashion. Kafka Connect makes it simple to quickly define connectors that move large collections of data into and out of Kafka. Source connectors can ingest entire databases or collect metrics from all your application servers into Kafka topics, making the data available for stream processing with low latency. Sink connectors can deliver data from Kafka topics into secondary storage and query systems or into batch systems for offline analysis.

CSP Community Edition is shipped with many different Cloudera developed and publicly available connectors that each cover a specific use case for streaming data. All of these can be deployed, managed, and monitored using Streams Messaging Manager (SMM)

The following tutorial walks you through a simple use case where data is moved from a Kafka topic into a PostgreSQL database using the JDBC Sink connector, which is one of the Cloudera developed connectors shipped with CSP Community Edition.

Tip: Although this tutorial focuses on the JDBC Sink connector, the basic process of deploying and monitoring a connector using SMM is the same for all connectors.

Before you begin

Find out the name or ID of the Kafka and PostgreSQL container. You will need to pass the name or ID of the containers as parameters in some of the commands you will be running. The container name or ID can be listed using docker ps. For example:

```
docker ps -a --format '{{.ID}}\t{{.Names}}' --filter "name=kafka.(\d)" --fil
ter "name=postgres"
```

The Kafka container will either be called cspce-kafka-1 or cspce_kafka_1.

The PostgreSQL container will either be called cspce-postgresql-1 or cspce_postgresql_1.

Creating a database and table in PostgreSQL

In order to stream data using the JDBC Sink connector, a destination is required for that data. In this tutorial, the database and table is created using the PostgreSQL instance already deployed in CSP Community Edition.

Procedure

1. Open a new terminal window and run the following command:

```
docker exec -it [***POSTGRESQL CONTAINER NAME OR ID***] /bin/bash
```

2. Start the psql client.

```
psql
```

3. Create a database.

```
CREATE DATABASE csp;
```

4. Change the database owner to smm.

```
ALTER DATABASE csp OWNER TO smm;
```

5. Connect to the csp database.

```
\c csp
```

6. Create a table.

```
CREATE TABLE demo (Name VARCHAR ( 255 ), Role VARCHAR, Age INT);
```

7. Change the table owner to smm.

```
ALTER TABLE demo OWNER to smm;
```

Creating a topic and producing messages

Before you can deploy the JDBC Sink connector, you need a Kafka topic with some messages that you can use. The connector connects to this topic and streams the data from the topic into the PostgreSQL database. The topic will be created using SMM. Message production is done using the Kafka console producer.

Procedure

1. Access the SMM UI by entering the following in a browser window:

```
http://localhost:9991
```

2. Click (Topics) in the navigation sidebar.
3. Configure the topic as follows:
   - Topic Name: csp-connect
   - Partitions: 1
   - Availability: Low
   - Cleanup Policy: delete

   Important: You can only create topics using the Low preset or the Custom preset with REPLICATION FACTOR and MIN.INSYNC.REPLICAS set to 1. This is because CSP Community Edition has a single Kafka broker deployed. Additionally, the name of the topic can only contain ASCII alphanumeric characters (a-z, A-Z, 0-9), dots (.), underscores(_), and minus signs (-).
4. Click Save.
5. Verify that the topic was created.

   This can be done by typing csp-connect in the search field. If the topic was successfully created, it will be listed under Topics.
6. Open a terminal session and run the following command:

```
docker exec -it [***KAFKA CONTAINER NAME OR ID***] /bin/bash
```

7. Run the Kafka console producer.

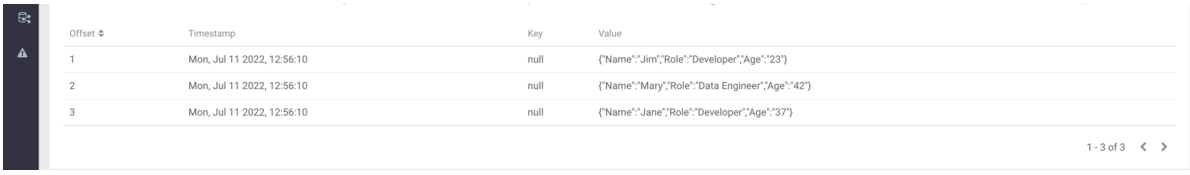
```
/opt/kafka/bin/kafka-console-producer.sh --bootstrap-server localhost:9094
 --topic csp-connect
```

8. Produce the following messages:

```
{"Name":"Jim","Role":"Developer","Age":"23"}
{"Name":"Mary","Role":"Data Engineer","Age":"42"}
{"Name":"Jane","Role":"Developer","Age":"37"}
```

   Notice how the schema of the messages matches the schema of the database table created in the previous section. This is required as otherwise the connector would not know where to put the data in the table. Ensure that each line in the example is produced as a single Kafka message.
9. Using the SMM UI, verify that the messages were produced.
   a) Go Topics and find the csp-connect topic.
   b) Click (Profile) and go to Data Explorer.

   If message production was successful, you should see the messages produced on the UI.



Offset	Timestamp	Key	Value
1	Mon, Jul 11 2022, 12:56:10	null	{"Name":"Jim","Role":"Developer","Age":"23"}
2	Mon, Jul 11 2022, 12:56:10	null	{"Name":"Mary","Role":"Data Engineer","Age":"42"}
3	Mon, Jul 11 2022, 12:56:10	null	{"Name":"Jane","Role":"Developer","Age":"37"}

1 - 3 of 3

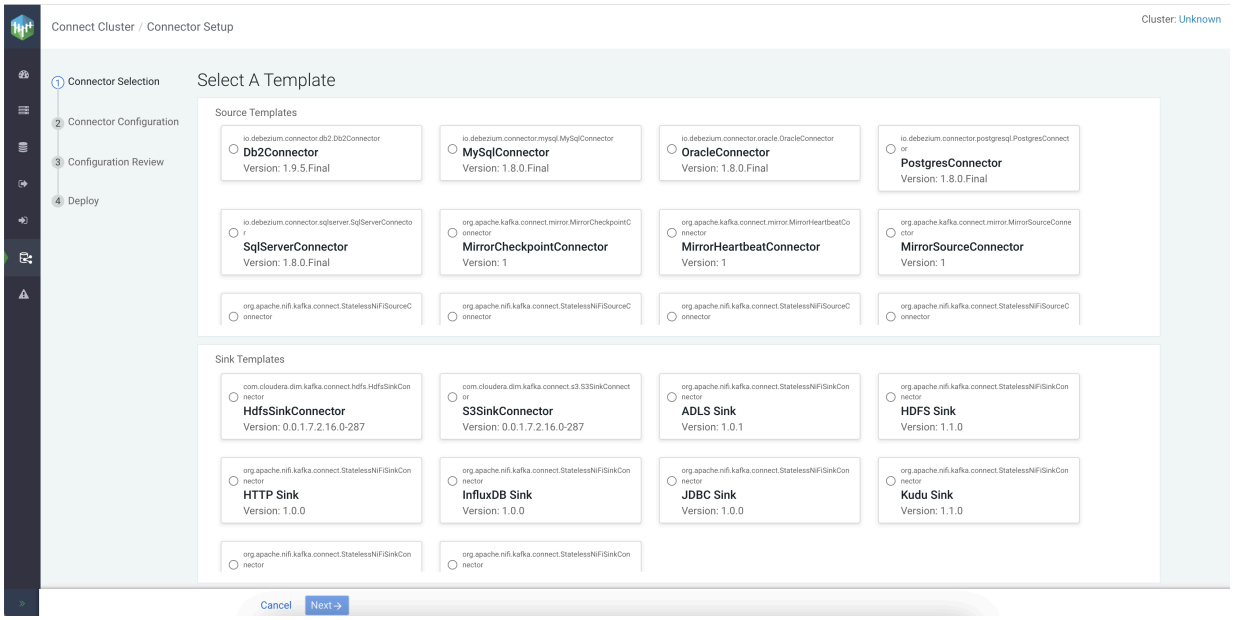
Deploying the connector

After both the topic and database is set up, it's time to deploy the JDBC Sink connector using the SMM UI.

Procedure

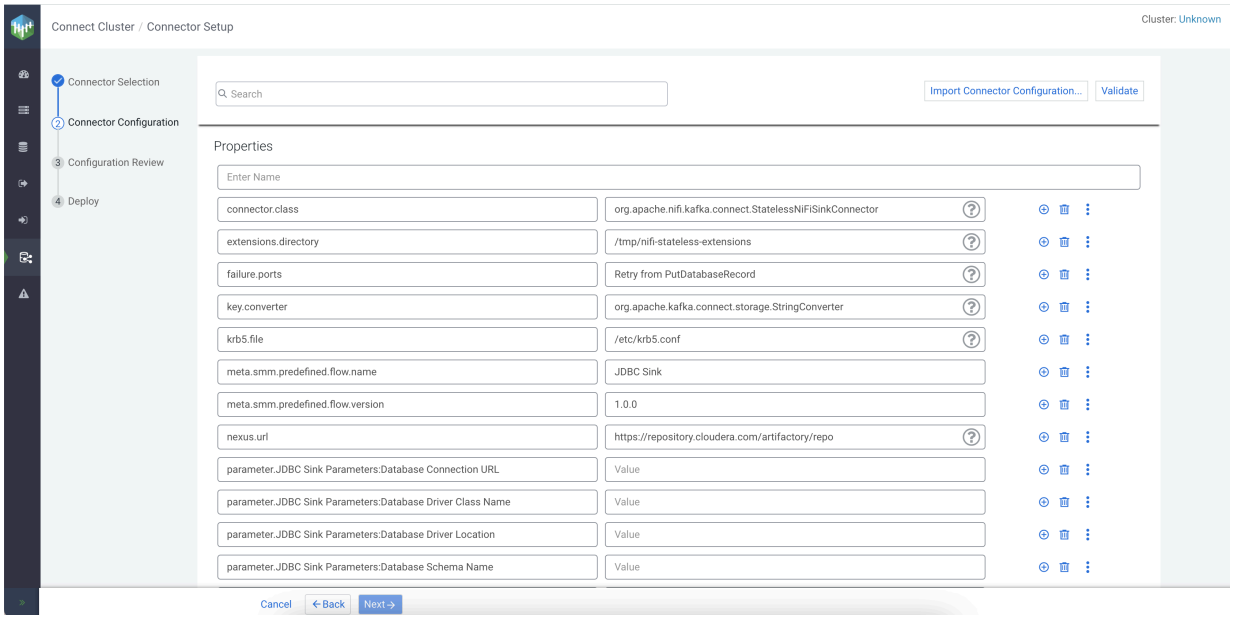
1. Click (Connect) on the navigation sidebar of the SMM UI.
2. Click +New Connector.

   Clicking +New Connector opens the **Select A Template** page. This page contains all connectors available for deployment in the form of selectable cards.



3. Navigate to the **Sink templates** section and select JDBC Sink.

   After a connector is selected, you are presented with the **Connector Configuration** page.



Most connectors are shipped with a default configuration template to ease configuration. If a template is available for a specific connector, the property keys and values are automatically populated when you select the connector. The properties and values included in the templates depend on the selected connector.

In addition, the page has various features and options that are designed to help you with configuration. For example, hovering over (Help), found next to some property values, displays information about that property.

For more information on what each UI option and icon does, see *Monitoring Kafka Connect Using Streams Messaging Manager* in the *Related Information* section at the bottom of this page.

4. Click Import Connector Configuration....
5. Copy and paste the following configuration in the Import Connector Config… dialog.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSinkConnec
tor",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "failure.ports": "Retry from PutDatabaseRecord",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "meta.smm.predefined.flow.name": "JDBC Sink",
 "meta.smm.predefined.flow.version": "1.0.0",
 "name": "sink-demo",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "parameter.JDBC Sink Parameters:Database Connection URL": "jdbc:postgresq
l://postgresql:5432/csp",
 "parameter.JDBC Sink Parameters:Database Driver Class Name": "org.postgre
sql.Driver",
 "parameter.JDBC Sink Parameters:Database Driver Location": "/opt/connect/
plugin/libs/debezium-connector-postgres/postgresql-connector-java.jar",
 "parameter.JDBC Sink Parameters:Database Table Name": "demo",
 "parameter.JDBC Sink Parameters:Database Type": "PostgreSQL",
 "parameter.JDBC Sink Parameters:Database User Name": "smm",
 "parameter.JDBC Sink Parameters:Database User Password": "cloudera",
 "parameter.JDBC Sink Parameters:Kafka Message Data Format": "JSON",
 "parameter.JDBC Sink Parameters:Schema Access Strategy": "Infer Schema",
 "tasks.max": "1",
 "topics": "csp-connect",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConver
ter",
 "working.directory": "/tmp/nifi-stateless-working"
}
```

Notice that Schema Access Strategy is set to Infer Schema. This tells the connector to figure out the schema of the data based on the data in the topic. Based on the format of the data, the connector will know how to match the data in the topic with the table columns. Note, however, that the connector is capable of retrieving the schema by other means, for example from Schema Registry. For more information regarding Schema Access Strategy as well as the other properties included in this example, see *JDBC Sink connector reference* in the *Related Information* section at the bottom of this page.

6. Click Import.
7. Click Validate.
8. Click Next.
9. Click Deploy.

   A pop-up window appears notifying you that deployment was successful. At this point, the connector is running. You can click View Connector Profile to close the dialog.
10. Query the database.

    After the connector is deployed, switch over to the terminal session you used to set up the database and run the following command:

```
SELECT * FROM public.demo;
```

Note: You might need to wait a few minutes until the entries become visible.

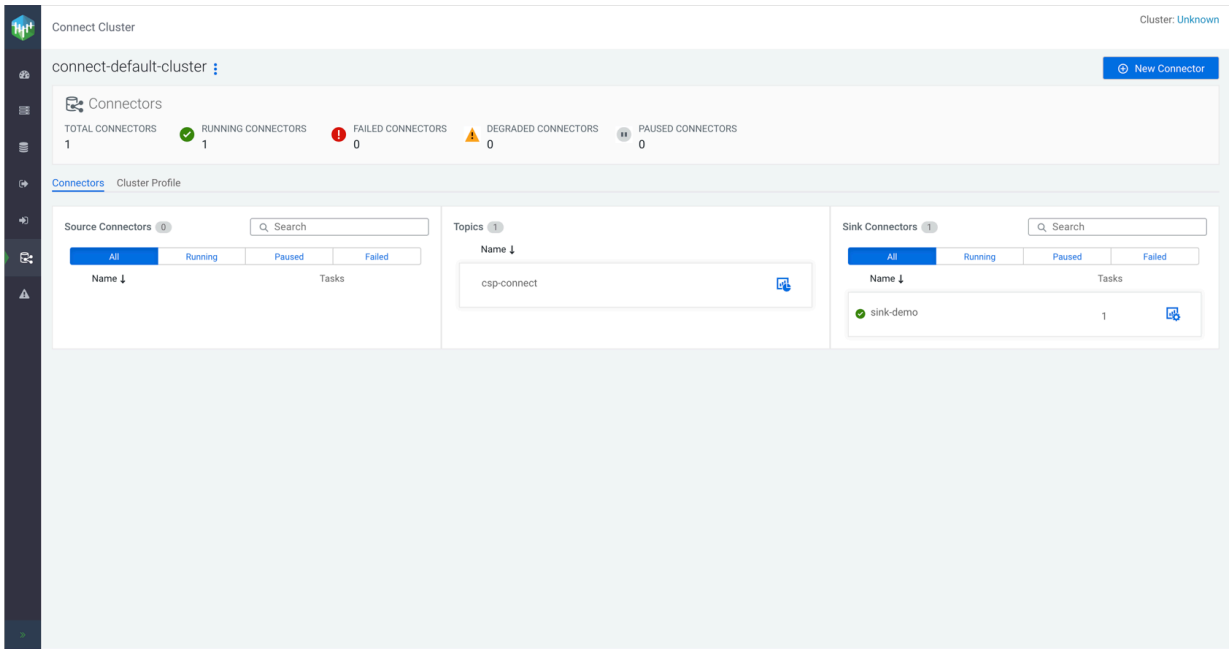
Monitoring and managing the connector

Once the connector is deployed, you can monitor its activity using the SMM UI.

Procedure

1. In the SMM UI click (Connect) on the navigation sidebar.

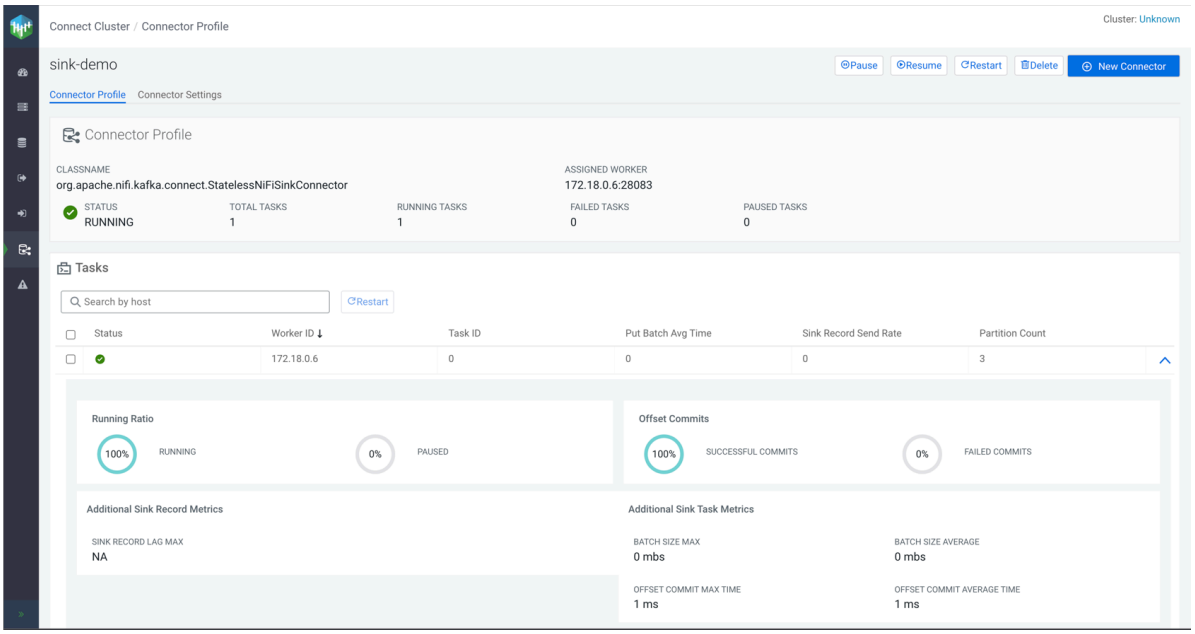
   Clicking on opens the Connect Cluster page with the Connectors tab open. This page provides a high level overview and metrics of all your connectors as well as the connect cluster.



2. Click (Connector Profile) next to the connector's name.

   The **Connector Profile** page enables you to monitor details of the connector and its tasks as well as its configuration. The page consists of two tabs Connector Profile and Connector Settings. The following points give a quick introduction of the UI and its features. Feel free to experiment and explore.

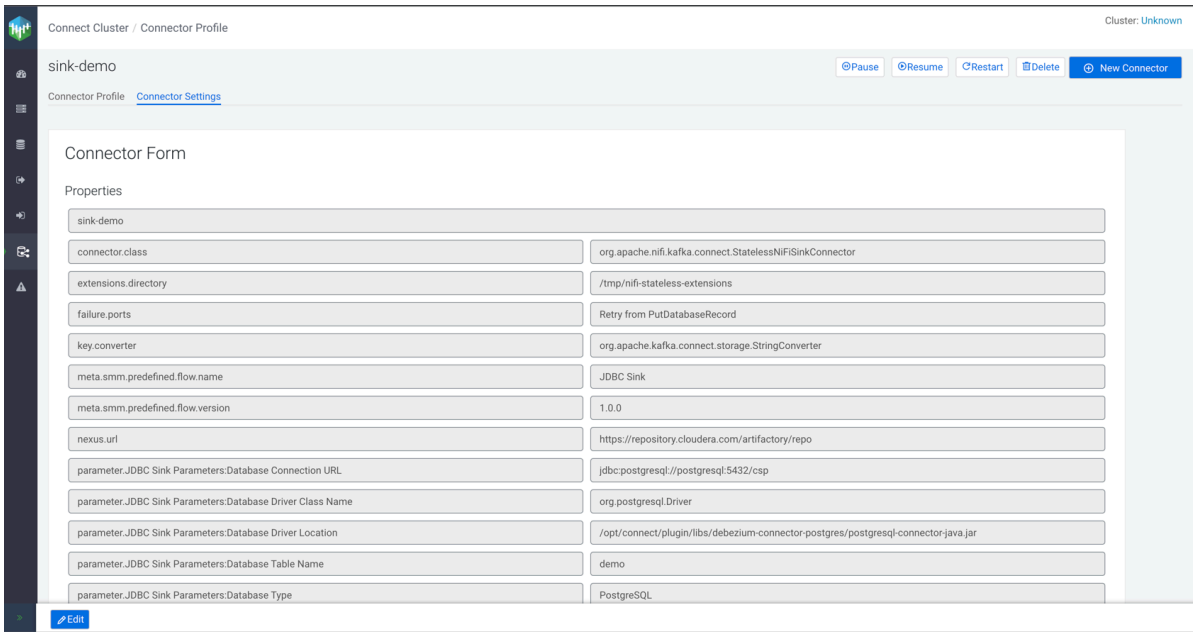
For Connector Profile



- The **Connector Profile** section provides you with details regarding the **Classname**, **Assigned Worker**, **Status**, **Total Tasks**, **Running Tasks**, **Failed Tasks**, and **Paused Tasks.**
- Using the buttons in the top right-hand corner you can pause, resume, restart, or delete the connector.
- In the **Tasks** section, you can view and monitor **Status**, **Worker ID**, **Task ID**, and various other details regarding connector tasks. Clicking next to a task displays detailed information and metrics about the selected task. In addition to viewing status and metrics, the **Tasks** section also allows you to restart

a particular task. This can be done by selecting the task you want to restart and then clicking the Restart option found within the **Tasks** section.

For Connector Settings



- The **Connector Settings** tab enables you to view and edit the configuration of the connector.
- Clicking Edit in the bottom left-hand corner allows you to reconfigure the connector.
- Using the buttons in the top right-hand corner you can pause, resume, restart, or delete the connector.

What to do next

Now that you are familiar with the basics of connector deployment and monitoring, Cloudera recommends that you explore and learn about the various other connectors that are included with CSP Community Edition. For more information on each of the connectors, see *Connectors* in the related information section.

Related Information

Cloudera Runtime - Connectors

Cloudera Runtime - JDBC Sink connector

Cloudera Runtime - JDBC Sink connector reference

Cloudera Runtime - Monitoring Kafka Connect Using Streams Messaging Manager

Getting Started with Streams Messaging Manager Alert Policies

Learn how to set up Alert Policies and Notifiers, how to view generated alerts, and how to disable an Alert Policy in Streams Messaging Manager (SMM).

About this task

Alert Policies in SMM are user configured alerts that automatically trigger when configured conditions are met. An alert contains the details of the policy including the alert message and the conditions that triggered the alert. Once an alert is triggered, a notification is sent out by SMM using the Notifier associated with the Alert Policy. You can use these alerts and notifications to monitor the health of Kafka, as well as to identify and troubleshoot problems.

For example, assume that you want to monitor your Kafka topics to ensure that data production and consumption is happening without interruptions. To do this you could set up an Alert Policy that triggers if the bytes consumed from

or produced to your topics fall below a specified threshold. When the alert is triggered, SMM sends out a notification using the Notifier associated with the Alert Policy immediately notifying you of a potential issue.

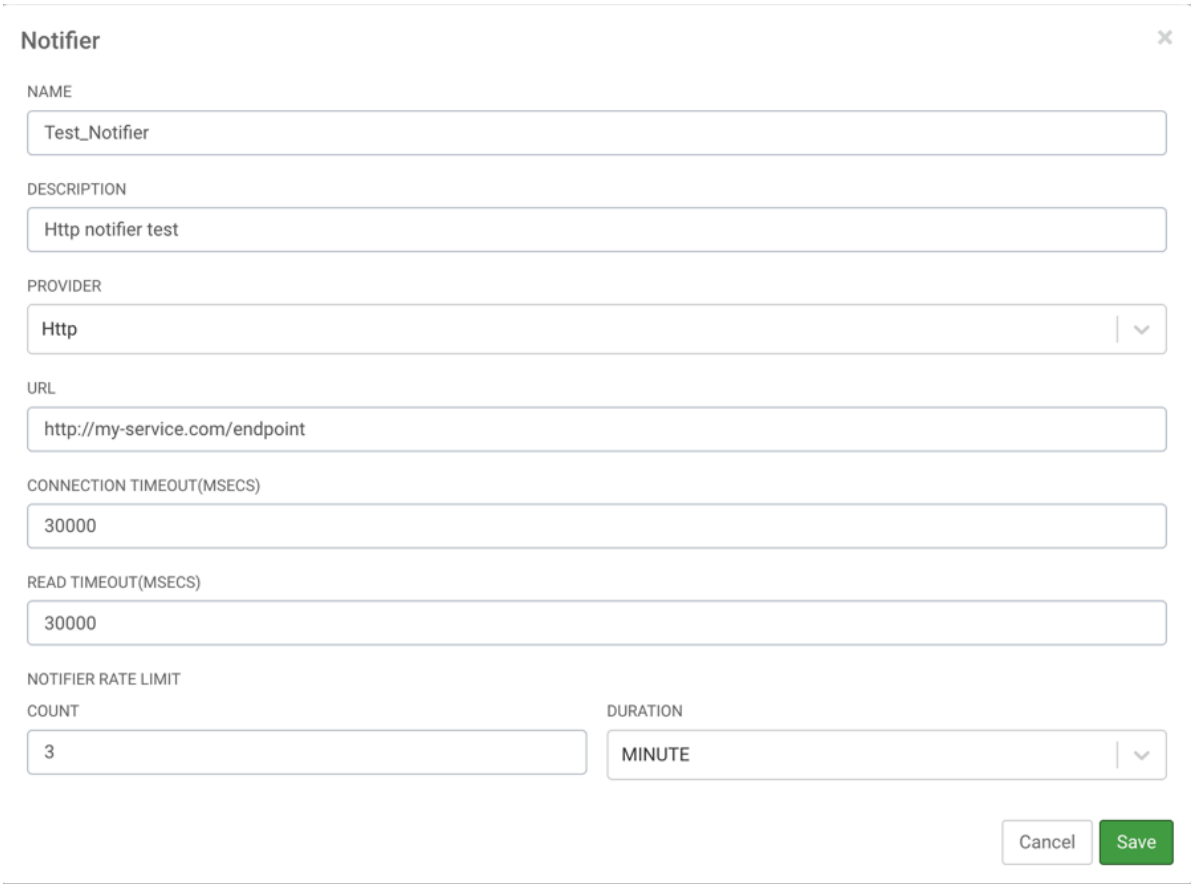
The following steps walk you through how to set up a simple Alert Policy that uses an HTTP Notifier, how to view the generated alerts within SMM, and how to disable an Alert Policy.

Procedure

1. Access the SMM UI by entering the following in a browser window:

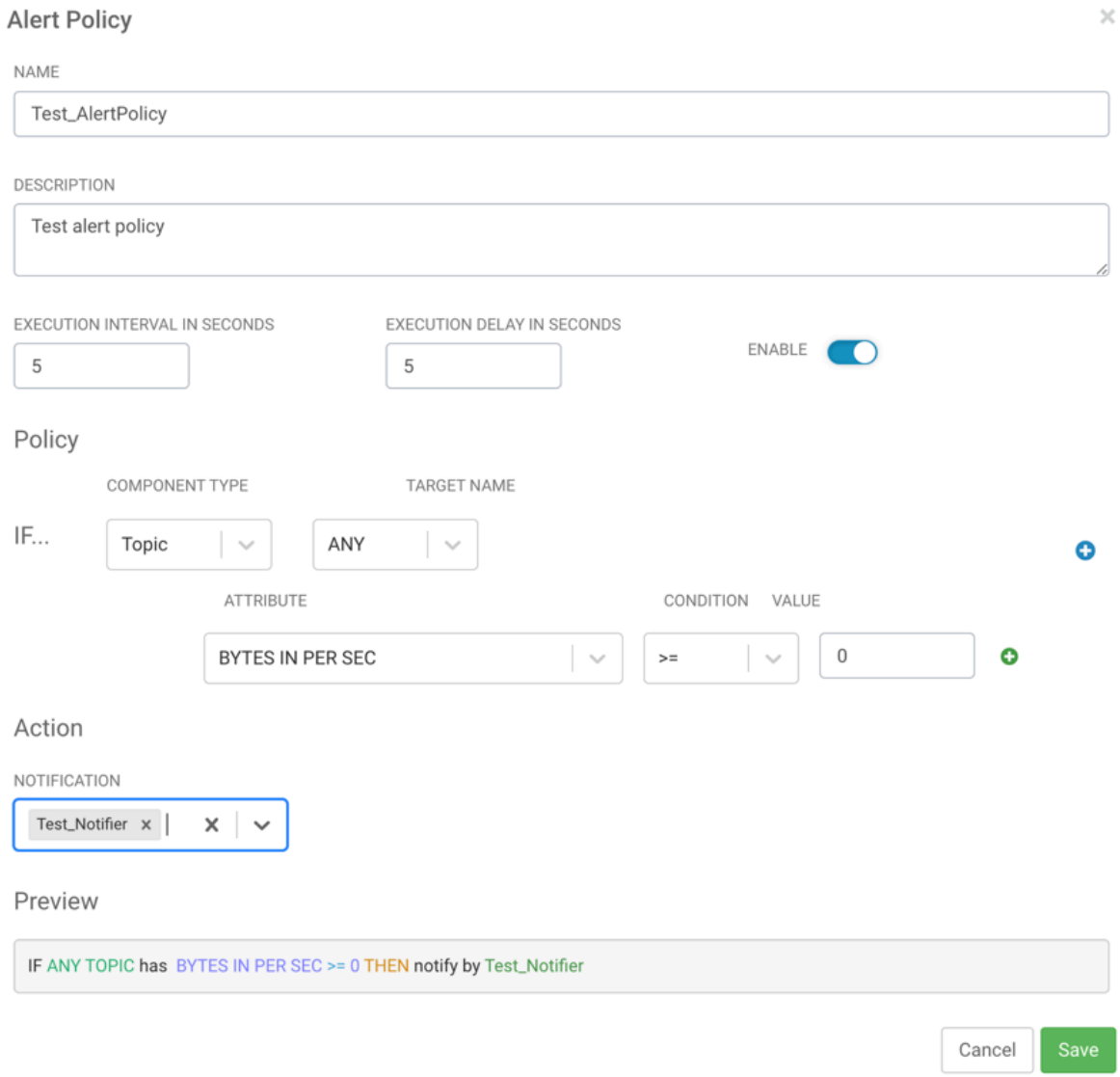
```
http://localhost:9991
```

2. Click (Alerts) on the navigation sidebar.
3. Create a notifier.
   a) Go to NOTIFIERS.
   b) Click ADD NEW.
   c) Configure the notifier as follows:



   Take note of the URL property. This property specifies the URL where this notifier sends alerts. You do not need to add the URL of an actual service to create and test Alert Policies. As a result, for this tutorial, you can set this property to a placeholder value. For example http://my-service.com/endpoint. In a production environment however, the URL you enter would be an endpoint of a target service that can consume and process the alerts that SMM sends.
   d) Click Save.

4. Create an Alert Policy.
   a) Go to ALERT POLICIES.
   b) Click ADD NEW.
   c) Configure the Alert policy as follows:



   d) Click Save.
5. View alerts.

   Go to History. Wait a few seconds and refresh the page, alerts should appear on the page.
6. Disable the Alert Policy.
   a) Go to Alert Policies.
   b) Find the alert policy that you want to disable, and click the toggle button beside the alert policy.

What to do next

To learn more about Alert Policies and Notifiers, see *Managing Alert Policies Using Streams Messaging Manager*.

Related Information

Managing Alert Policies Using Streams Messaging Manager

Setting up connectors that use JDBC drivers

The Debezium and JDBC Kafka Connect connectors shipped with CSP Community Edition require a JDBC driver to function. However, by default only the PostgreSQL JDBC driver is available for use. If you want to deploy a connector that requires a different JDBC driver to function, you must copy the driver to the Kafka Connect container before you deploy the connector.

About this task

The following Kafka Connect connectors require a JDBC driver that does not come shipped with CSP Community Edition:

- Debezium Db2 Source
- Debezium MySQL Source
- Debezium Oracle Source
- Debezium SQL Server Source
- JDBC Source
- JDBC Sink

Before you can deploy any of these connectors you must copy the associated JDBC driver to the Kafka Connect Container. For the Debezium connectors, the location where you copy the driver JAR is fixed. Additionally, the Debezium connectors require that the JAR file is deployed within the container using a specific file name.

Important: The PostgreSQL JDBC driver comes bundled with CSP Community Edition. It is deployed at the following location in the Kafka Connect container.

```
/opt/connect/plugin/libs/debezium-connector-postgres/postgresql-conn
ector-java.jar
```

This means that you do not need to complete the following steps if:

- You are deploying the Debezium PostgreSQL Source connector.
- You are deploying the JDBC Source or Sink connectors and use them to source or sink data from a PostgreSQL database.

Before you begin

Find out the name or ID of the Kafka Connect container. You will need to pass the container name or ID as a parameter in some of the commands you will be running.The container name and ID can be listed using docker ps. For example:

```
docker ps -a --format '{{.ID}}\t{{.Names}}' --filter "name=kafka-connect"
```

The Kafka Connect container will either be called cspce-kafka-connect-1 or cspce_kafka-connect_1.

Steps

Debezium Db2 Source

1. Download the Db2 JDBC driver.

   The JDBC driver can be downloaded from *Maven Central*. Cloudera recommends that you use version 11.5.0.0.
2. Navigate to the directory where the driver was extracted.
3. Copy the driver to the Kafka Connect container and rename it.

   ```
   docker cp ./jcc-[***VERSION***].jar [***KAFKA CONNECT
    CONTAINER NAME OR ID***]:/opt/connect/plugin/libs/debezium-c
   onnector-db2/db2-connector-java.jar
   ```

4. Log in to the Kafka Connect container as the root user.

```
docker exec -it --user root [***KAFKA CONNECT CONTAINER ID OR
 NAME***] /bin/bash
```

5. Change the owner of the driver to the kafka user.

```
chown kafka:cloudera /opt/connect/plugin/libs/debezium-conne
ctor-db2/db2-connector-java.jar
```

Debezium MySQL Source

1. Download the MySQL JDBC driver.

   The JDBC driver can be downloaded from the *MySQL Community Downloads* page. Download the platform independent version. Cloudera recommends that you use version 8.0.27.

2. Extract the files from the downloaded archive.

   You can use the tar command or any other archive manager to extract the archive. For example:

```
tar -xf mysql-connector-java-[***VERSION***].tar.gz
```

3. Navigate to the directory where the driver was extracted.

```
cd mysql-connector-java-[***VERSION***]
```

4. Copy the driver to the Kafka Connect container and rename it.

```
docker cp ./mysql-connector-java-[***VERSION***].jar [***KAFKA
 CONNECT CONTAINER NAME OR ID***]:/opt/connect/plugin/libs/de
bezium-connector-mysql/mysql-connector-java.jar
```

5. Log in to the Kafka Connect container as the root user.

```
docker exec -it --user root [***KAFKA CONNECT CONTAINER ID OR
 NAME***] /bin/bash
```

6. Change the owner of the driver to the kafka user.

```
chown kafka:cloudera /opt/connect/plugin/libs/debezium-conne
ctor-mysql/mysql-connector-java.jar
```

Debezium Oracle Source

1. Download the Oracle JDBC driver.

   The JDBC driver can be downloaded from *Maven Central*. Cloudera recommends that you use version 12.2.0.1, 19.8.0.0, or 21.1.0.0.

2. Navigate to the directory where the driver was downloaded.
3. Copy the driver to the Kafka Connect container and rename it.

```
docker cp ./ojdbc8-[***VERSION***].jar [***KAFKA CONNECT
 CONTAINER NAME OR ID***]:opt/connect/plugin/libs/debezium-co
nnector-oracle/oracle-connector-java.jar
```

4. Log in to the Kafka Connect container as the root user.

```
docker exec -it --user root [***KAFKA CONNECT CONTAINER ID OR
 NAME***] /bin/bash
```

5. Change the owner of the driver to the kafka user.

```
chown kafka:cloudera /opt/connect/plugin/libs/debezium-conne
ctor-oracle/oracle-connector-java.jar
```

Debezium SQL Server source

1. Download the SQL Server JDBC driver.

   The JDBC driver can be downloaded from the *Microsoft JDBC Driver for SQL Server downloads* page. Cloudera recommends that you use version 7.2.2.jre8. Download the zipped tar file, do not download the executable version.
2. Extract the files from the downloaded archive.

   You can use the tar command or any other archive manager to extract the archive. For example:

```
tar -xf sqljdbc_[***VERSION***]_enu.tar.gz
```

3. Navigate to the directory where the driver was extracted.

```
cd sqljdbc_[***VERSION***]/enu
```

4. Copy the driver to the Kafka Connect container and rename it.

```
docker cp ./mssql-jdbc-[***VERSION***].jre8.jar [***KAFKA
 CONNECT CONTAINER NAME OR ID***]:/opt/connect/plugin/libs/de
bezium-connector-sqlserver/sqlserver-connector-java.jar
```

5. Log in to the Kafka Connect container as the root user.

```
docker exec -it --user root [***KAFKA CONNECT CONTAINER ID OR
 NAME***] /bin/bash
```

6. Change the owner of the driver to the kafka user.

```
chown kafka:cloudera /opt/connect/plugin/libs/debezium-conne
ctor-sqlserver/sqlserver-connector-java.jar
```

JDBC Source and Sink

1. Download the driver.

   The driver you need to download will depend on your use case and how you plan on using the JDBC Source or Sink connector. For example. if you want the connector to connect to a MySQL database, you need to download and deploy the MySQL JDBC driver.
2. Extract the files from the downloaded archive. You can use the tar command or any other archive manager to extract the archive. For example:

```
tar -xf [***PATH TO JDBC DRIVER ARCHIVE***]
```

   Depending on the driver you downloaded, this step might not be required.
3. Copy the driver to the Kafka connect container. For example:

```
docker cp [***PATH TO JDBC DRIVER JAR***] [***KAFKA CONNECT
 CONTAINER NAME OR ID***]:/opt/connect/plugin/libs/
```

   You can copy the driver to any location within the container. Cloudera recommends that you copy it to a directory that has no other JAR files. Additionally, ensure that you take note of the path as you must provide it in the connector's configuration during connector deployment.

4. Change the owner of the driver to the kafka user.

```
chown kafka:cloudera [***PATH TO DRIVER JAR***]
```

Results

The selected drivers are copied within the Kafka Connect container. Deploying the Debezium, JDBC Source, and JDBC Sink connectors is now possible.

What to do next

Deploy your selected connector. For more information on how to deploy the connectors, their configuration, and features, see the Related Information section at the bottom of this page.

Related Information

Debezium - Debezium connector for Db2
Debezium - Debezium connector for MySQL
Debezium - Debezium Connector for Oracle
Debezium - Debezium connector for SQL Server
Cloudera Runtime - JDBC Sink connector
Cloudera Runtime - JDBC Source connector
MySQL Community Downloads
Maven Central - Db2 JDBC driver
Maven Central - Oracle JDBC driver
Microsoft JDBC Driver for SQL Server

Getting started with Streaming Analytics

Get started with Streaming Analytics in CSP Community Edition.

The Getting Started with SQL Stream Builder tutorial walks you through the basic usage and features of SQL Stream Builder powered by Apache Flink. Complete the Getting Started guide to learn how to use Streaming SQL Console, and how to query streams of data using SQL Stream Builder, and then you can try out the different supported connectors, or learn how to manage and monitor your SQL jobs.

Getting started with SQL Stream Builder

Get started with SQL Stream Builder in CSP Community Edition.

The Getting Started guide walks you through the following steps from which you can learn how to use SQL Stream Builder (SSB):

- How to access Streaming SQL Console
- How to create a project
- How to create a SQL job
- How to generate data to SSB
- How to create tables and views
- How to query data
- How to output the queried results using different solutions

1. Creating a project

As a first step to use SQL Stream Builder, learn how to create a project with git synchronization where you can create and manage jobs, resources and experience the SDLC of SSB applications.

After accessing Streaming SQL Console using http://localhost:18121 from your browser and signing in, you are redirected to the **Projects** page of Streaming SQL Console.

What are the benefits of Projects in SSB?

Providing Software Development Lifecycle (SDLC)

Projects allow you to group together resources for a task based on for example a business logic that you would like to solve with SSB. Resources are data sources, tables, user-defined functions, and Materialized Views that are needed for SQL jobs in a project.

Version control and backups with Git

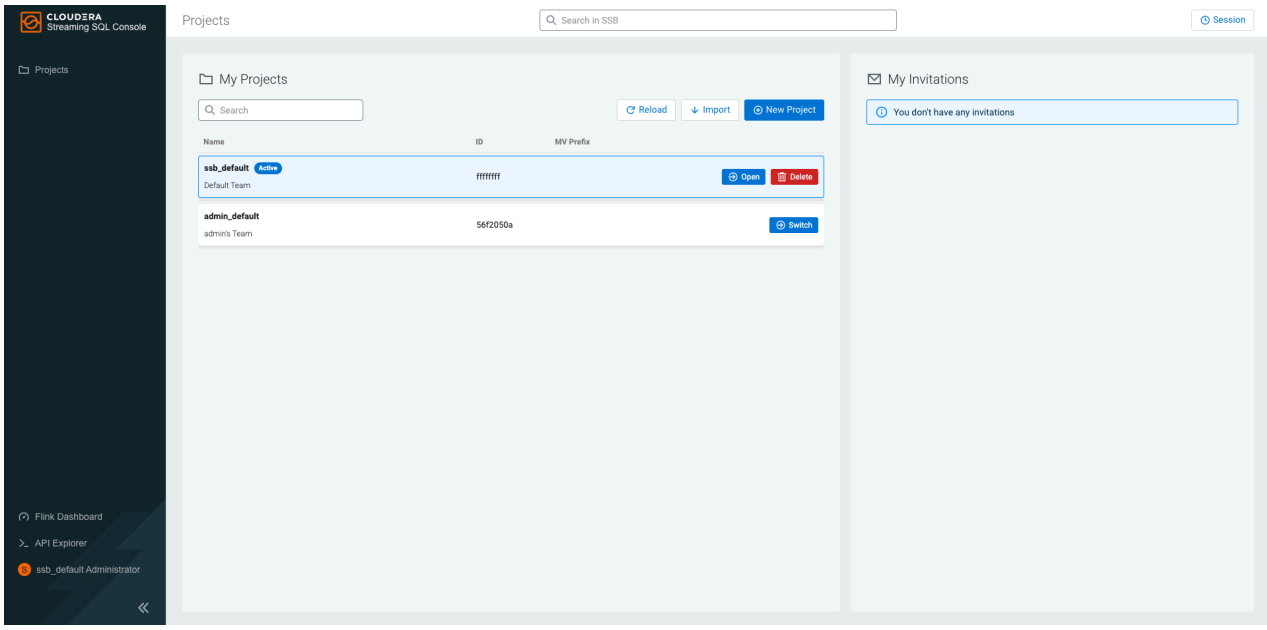
You can export and import the project to a Git repository, track versions using different branches and push your changes with commit messages as with any software project.

Enabling collaboration within a project

You can invite team members to your project to work together with the shared resources. The members you invite to a project can work with the same resources, and you can also assign an admin to the team who has the same privileges as the owner of the project.

Customizing environments with variables

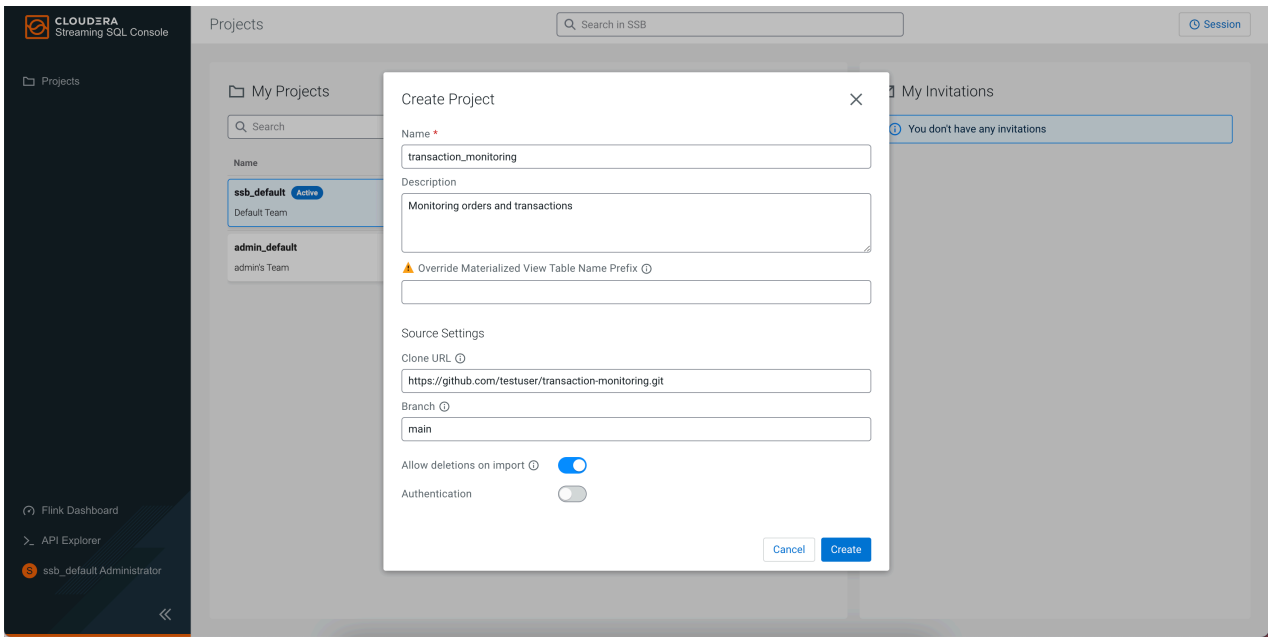
You can create environment files that contain project-specific configurations as key-value pairs that are used for substituting cluster specific information.



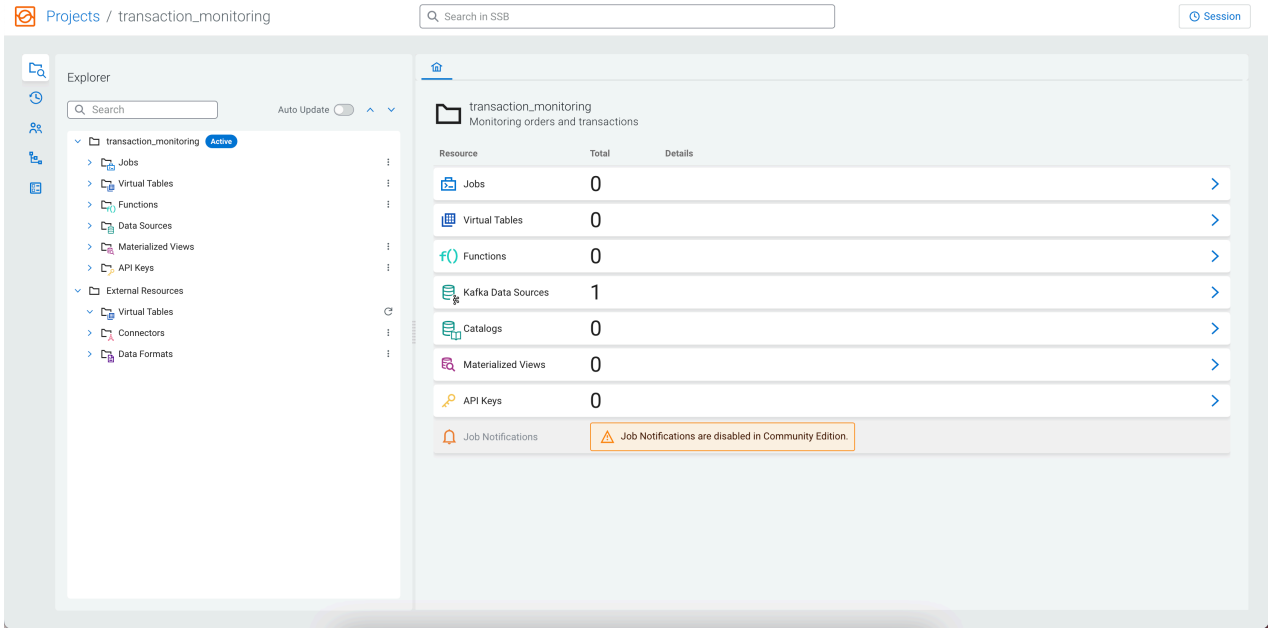
As an example, you can create a *transaction_monitoring* project using the Community Edition in your local environment, set up the resources such as tables, data sources and jobs, and then migrate the project using a Git repository to a CDP cluster or environment.

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 has its own project generated by default where members can be invited.

To create a project in Streaming SQL Console, you need to click on New Project on the **Projects** page and provide the Name of the project, add a description if necessary and set up the dedicated Git repository for synchronization. For example, you can name the project *transaction_monitoring*, add *Monitoring orders and transactions* as description, and use a dedicated Git repository with a main branch.



Click Create, and click Switch Switch Project to be redirected to the Explorer view of the created project.

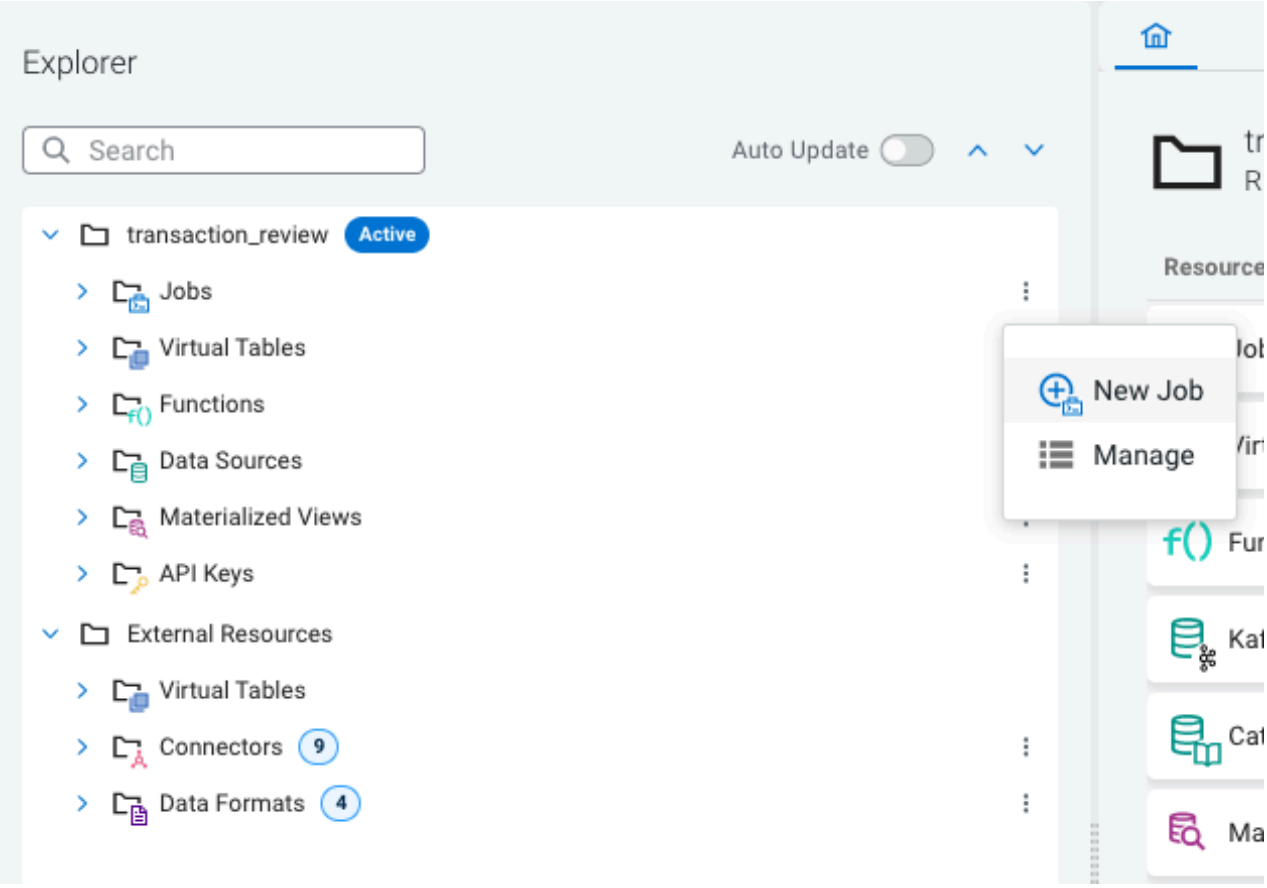


Within the created project, you can use the tree structure on the left side to review and manage the resources in a project. You can also use the home page of the project on the right side, and manage the resources of the project in a tab view.

2. Creating a job and table

When you created your project and set up Git for version control, learn how to create a job and a table that generates random data using the included Faker connector after accessing the Streaming SQL Console.

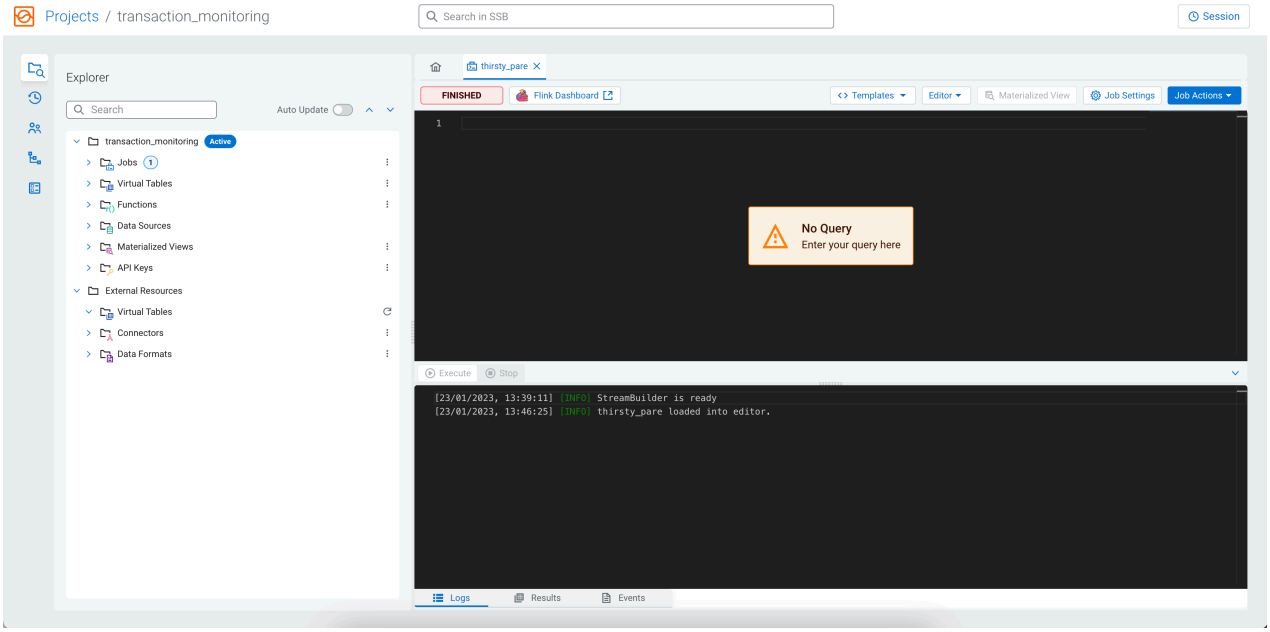
You can easily create a new job using the **Explorer** view of your project by clicking on the ⋮ button next to **Jobs** and select New Job.



This opens the **Create New Job** window where 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

After naming your job, click on the Create button to be redirected to the **SQL Editor** that enables you to create tables and query data, create Materialized Views, change job settings and manage jobs.

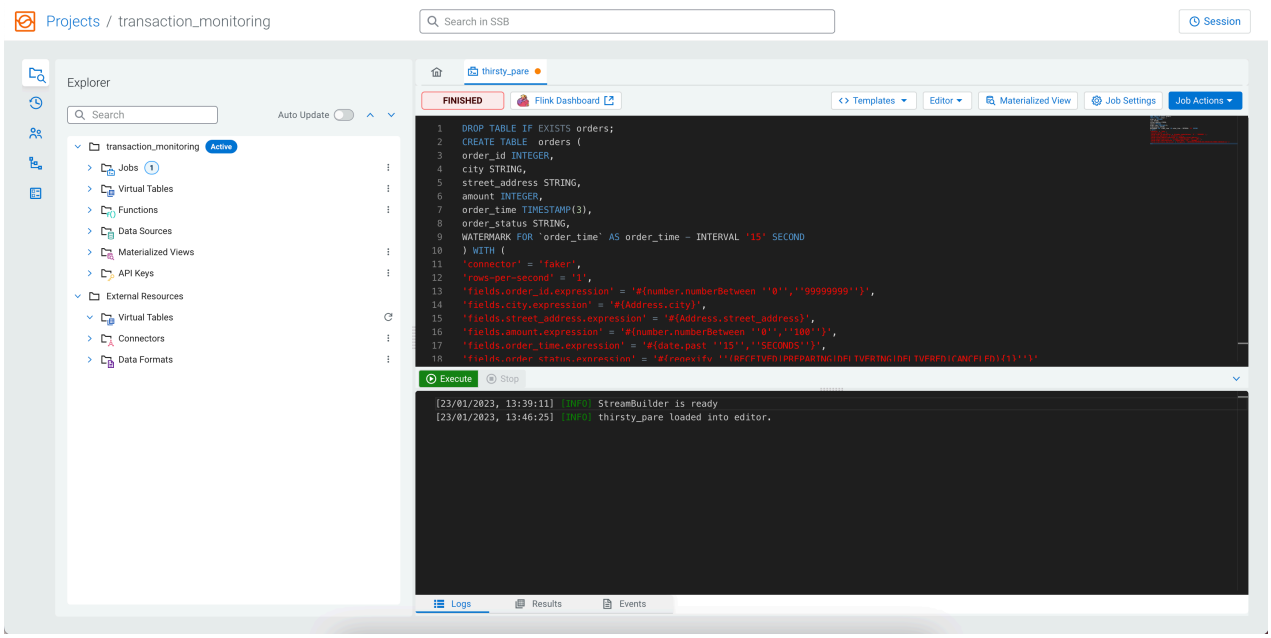


As a first step in executing a SQL query, you need to create a table. At this point, you can use the predefined templates under the **SQL Editor**.

In the following CREATE TABLE statement, the faker template is used, with the properties already filled out. You can copy and paste it to the **SQL Editor**.

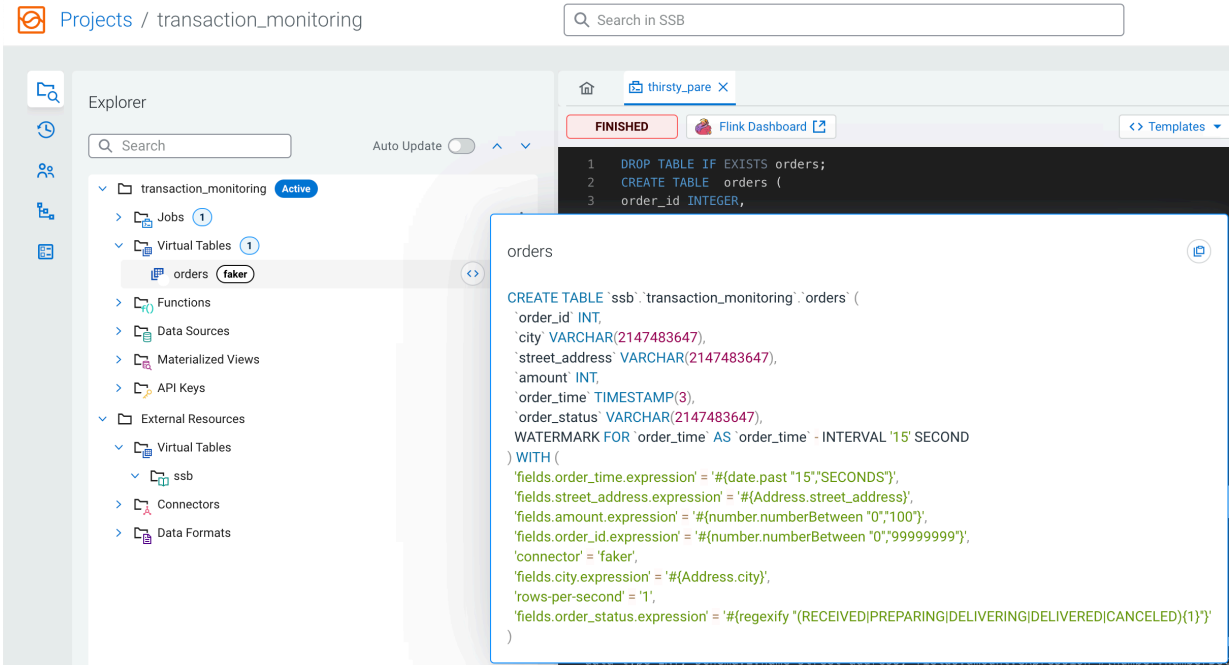
```
DROP TABLE IF EXISTS orders;
CREATE TABLE  orders (
    order_id INTEGER,
    city STRING,
    street_address STRING,
    amount INTEGER,
    order_time TIMESTAMP(3),
    order_status STRING,
    WATERMARK FOR `order_time` AS order_time - INTERVAL '15' SECOND
) WITH (
    'connector' = 'faker',
    'rows-per-second' = '1',
    'fields.order_id.expression' = '#{number.numberBetween ''0'',''99999999
''}',
    'fields.city.expression' = '#{Address.city}',
    'fields.street_address.expression' = '#{Address.street_address}',
    'fields.amount.expression' = '#{number.numberBetween ''0'',''100''}',
    'fields.order_time.expression' = '#{date.past ''15'',''SECONDS''}',
    'fields.order_status.expression' = '#{regexify ''(RECEIVED|PREPARING|DEL
IVERING|DELIVERED|CANCELED){1}''}'
);
```

To create the table, you need to click on the Execute button under the **SQL Editor**.

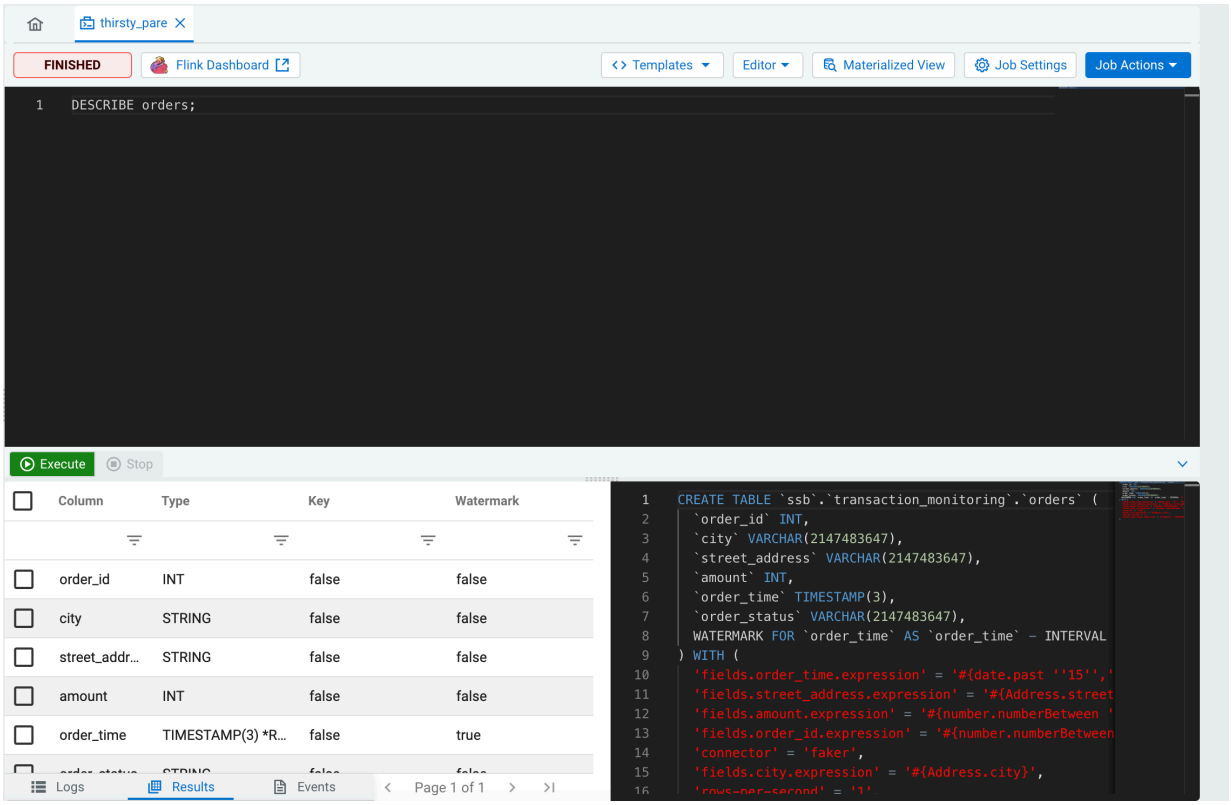


After executing the CREATE TABLE statement, there are two ways to view the created table:

- You can view the table under Virtual Tables of the project by clicking the arrow to expand the table properties:

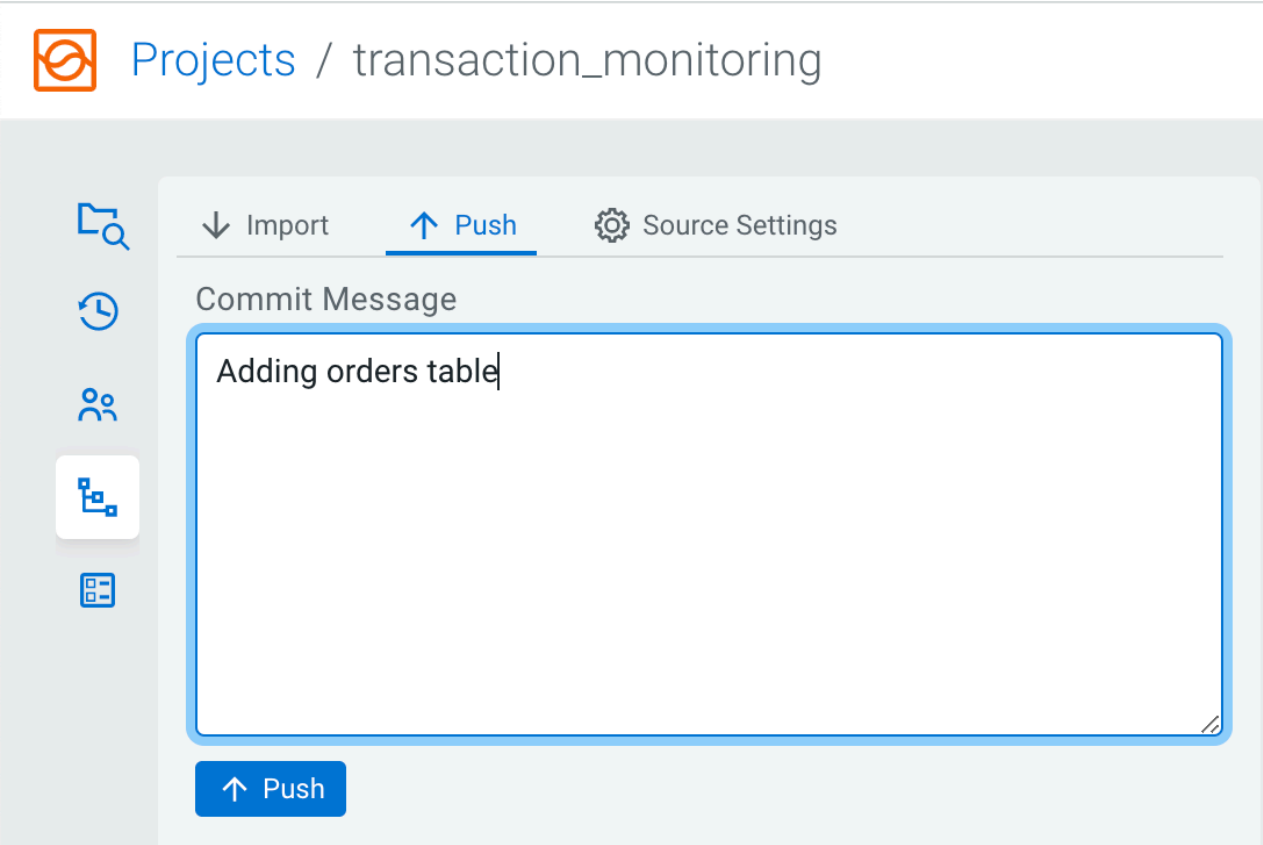


- In the SQL Editor window, you can replace the earlier statement by DESCRIBE orders;, and click Execute again:



Optional: Synchronizing your changes to Git

Now that you have created a job and a table, you can push your changes to your Git repository using the **Source Control** view and selecting the **Push** tab. Provide a commit message that describes the changes, and click Push to synchronize your changes to Git.



If someone else would work on this project in their local SSB Community Edition instance or SSB on a CDP cluster, they can use the Import button to pull the changes and have the latest version of the project in their SSB.

3. Running a SELECT query

After creating a table, learn how to run a SELECT query that outputs a sample of the resulting data in Streaming SQL Console.

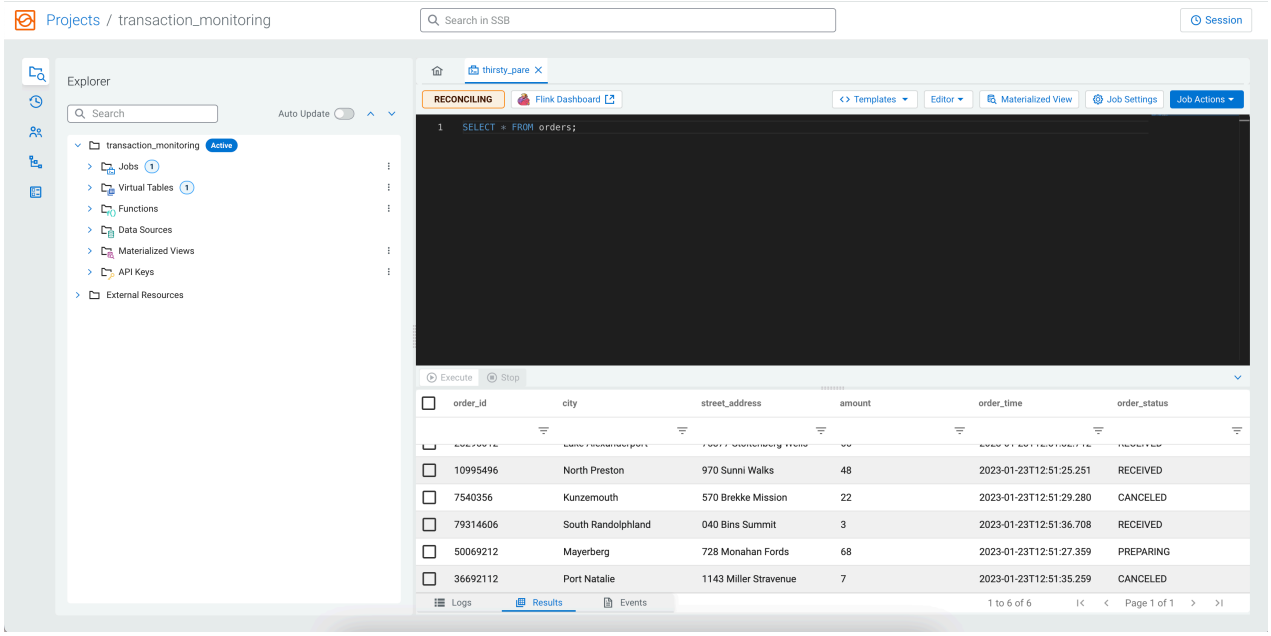
You have created your table using the Faker template that currently generates data. As a next step, you can run a SELECT query and output the results at the **SQL Editor** under the **Results tab**.

Copy and paste the following SQL query to the **SQL Editor**:

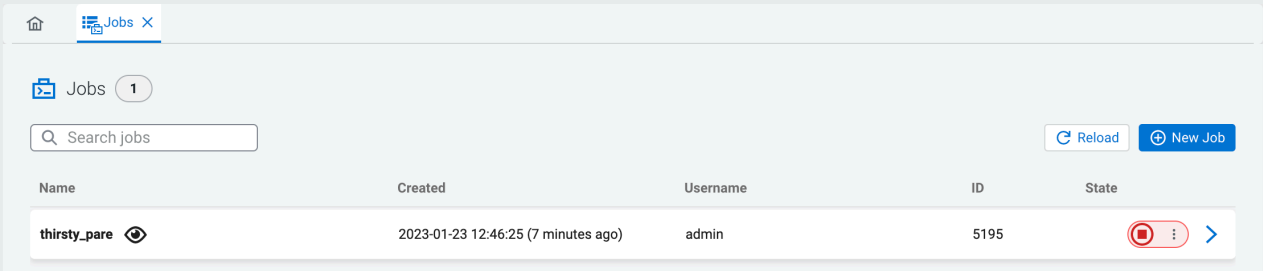
```
SELECT * FROM orders;
```

To execute the query, you need to click on the Execute button under the **SQL Editor**.

After a short amount of time, the samples of the result continuously show up under the **Results tab**.



When you execute a query, a Flink job is created in the background. You can see more details about the created Flink job by clicking on the Flink Dashboard button for the running job. You can also reload the SQL job into the **SQL Editor** to edit the job, or stop the job from the **Jobs** tab that lists all of the running and finished jobs in a project.



4. Creating and querying a view

Now that you know the basics of creating a table and querying data from a table, learn how to create a more complex query using views.

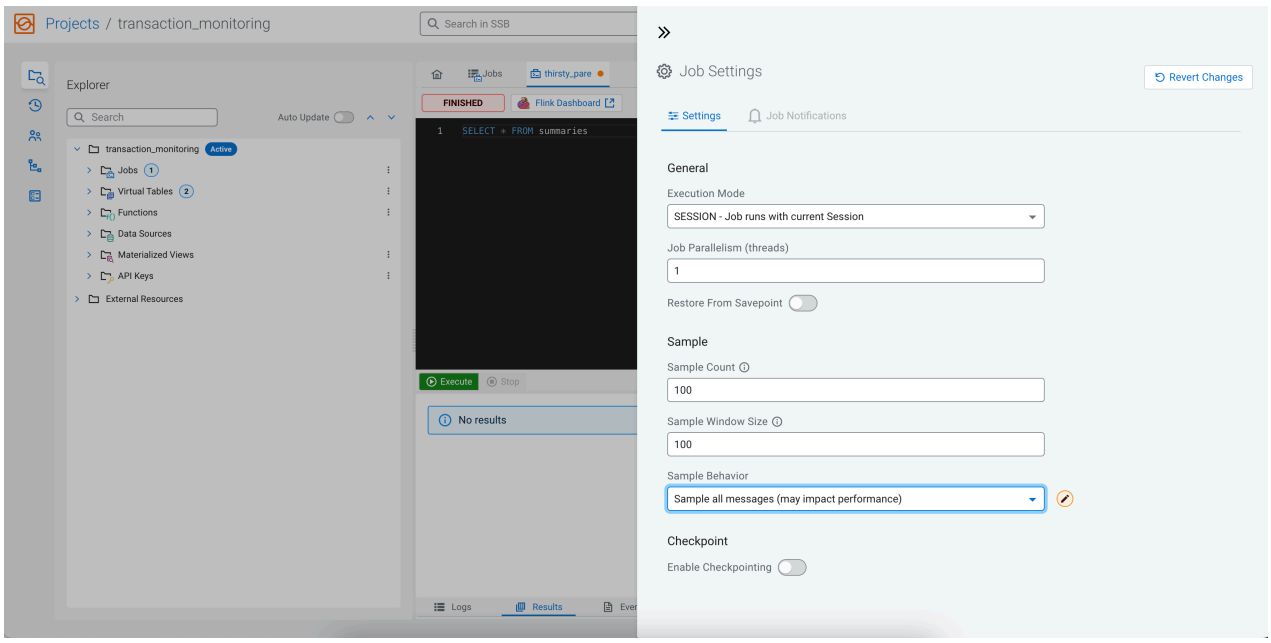
In the following example, you will use the columns from the created Faker table. In this derived computation that can be saved as a view, the orders are aggregated in half minute intervals based on their status.

Copy and paste the following CREATE VIEW statement to the **SQL Editor**:

```
DROP VIEW IF EXISTS summaries;
CREATE VIEW summaries AS
SELECT window_start, window_end, order_status, COUNT(*) as order_count, SUM(
amount) as total_amount
FROM TABLE(TUMBLE(TABLE orders, DESCRIPTOR(order_time), INTERVAL '20' SECOND
S))
GROUP BY window_start, window_end, order_status;
```

If your previous query is still running, click on the Stop button under the **SQL Editor** to stop the job. Afterwards, to create the view, you need to click on the Execute button under the **SQL Editor**.

When you execute a query, not all the records produced by the query are shown in the **Results** window. By default, 1 record is sampled every second to be displayed. Before selecting data from this view, you can change the **Sample Behavior** to Sample all messages. You need to click on the **Job Settings** button to open the settings that can be updated for a job. This way the **Results** window will contain all messages, up to the specified count.

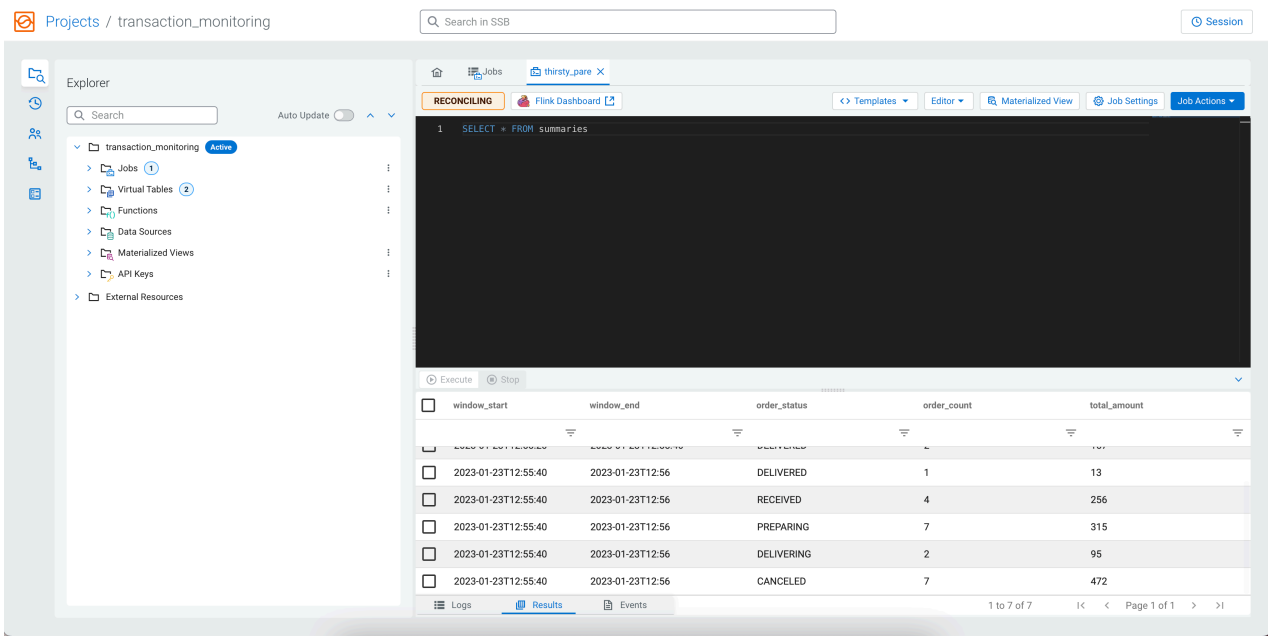


Add the following SQL query to the **SQL Editor** to run the query:

```
SELECT * FROM summaries
```

To execute the query, you need to click on the Execute button under the **SQL Editor**.

After a short amount of time, the samples of the aggregated result continuously show up under the Results tab. As it is an aggregated view, the results can appear slower after another.



5. Outputting results of SQL query

As you already know how to query tables and views, the next step is to write the queried results into a sink or create a Materialized View.

Adding a sink using a Local Kafka template

So far you have sampled the results to the Results tab of Streaming SQL Console, so the next step is to learn how to create a sink using one of the predefined templates to output the results of the executed SQL queries.

Previously, you created the summaries view, executed the SELECT * FROM summaries query, and examined its results on the Streaming SQL Console.

Now, the next step is to stream the aggregated records to an external system, such as Kafka. The Community Edition comes with a preconfigured Kafka container that can be selected from the predefined templates.

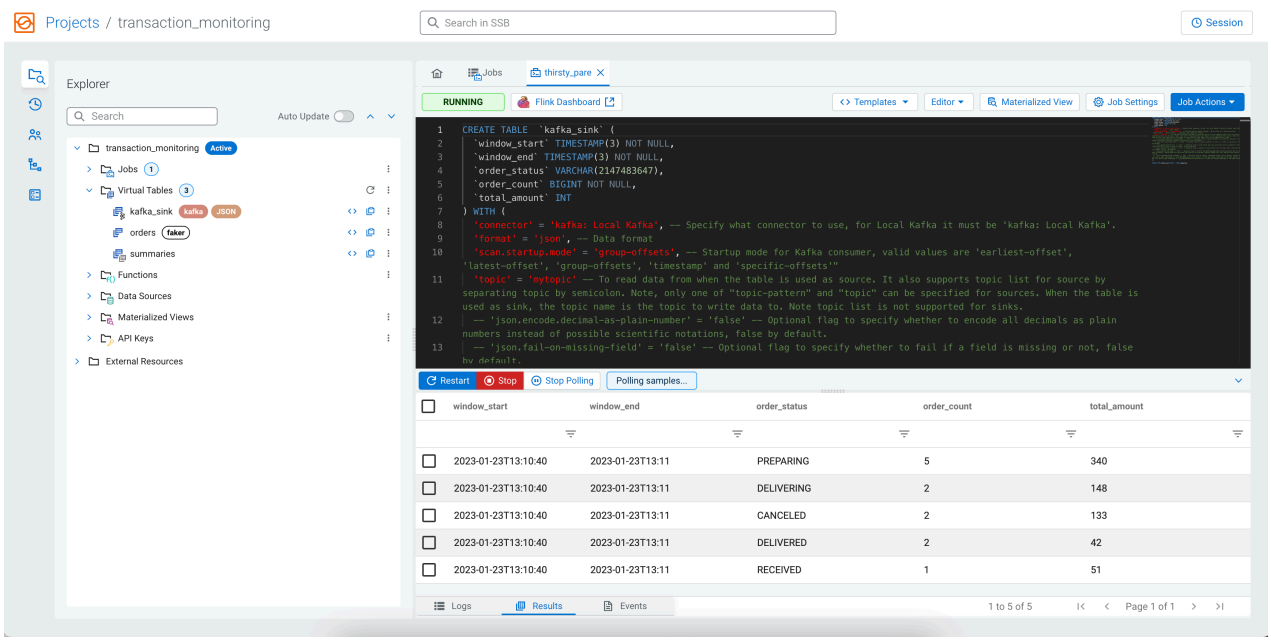
The created view has a complex schema. It would be tedious to describe it, and manually create a Kafka table with a matching schema. Luckily, SQL Stream Builder can do this task automatically.

As you can add more statements in one SQL window, you can create the Kafka table and add a SELECT query at the same time.

Add the following SQL query to the SQL window, but do not execute it. At this point the statement is only used to match the schema from summaries table with the schema of the kafka_sink that will be created at the next step:

```
SELECT * FROM summaries;
```

As the *kafka_sink* does not exist yet, you need to import the CREATE TABLE statement to the SQL window by selecting Templates > local-kafka > json . SSB automatically inserts the appropriate CREATE TABLE statement with the schema retrieved from the summaries table. Before executing the statement, you must fill out the topic in the connector option. This means you need to replace '...' in the template with 'mytopic' as it is shown in the following example:



Click on the Execute button to run the script, both statements will be executed.

Add the SELECT query after the CREATE TABLE statement:

```
INSERT INTO kafka_sink SELECT * FROM summaries;
```

Optional: Creating an Environment configuration for your project

Before moving forward, you can create Environment variables that can be used to store environment-specific configurations. You can add key-value pairs to create placeholders for the following properties:

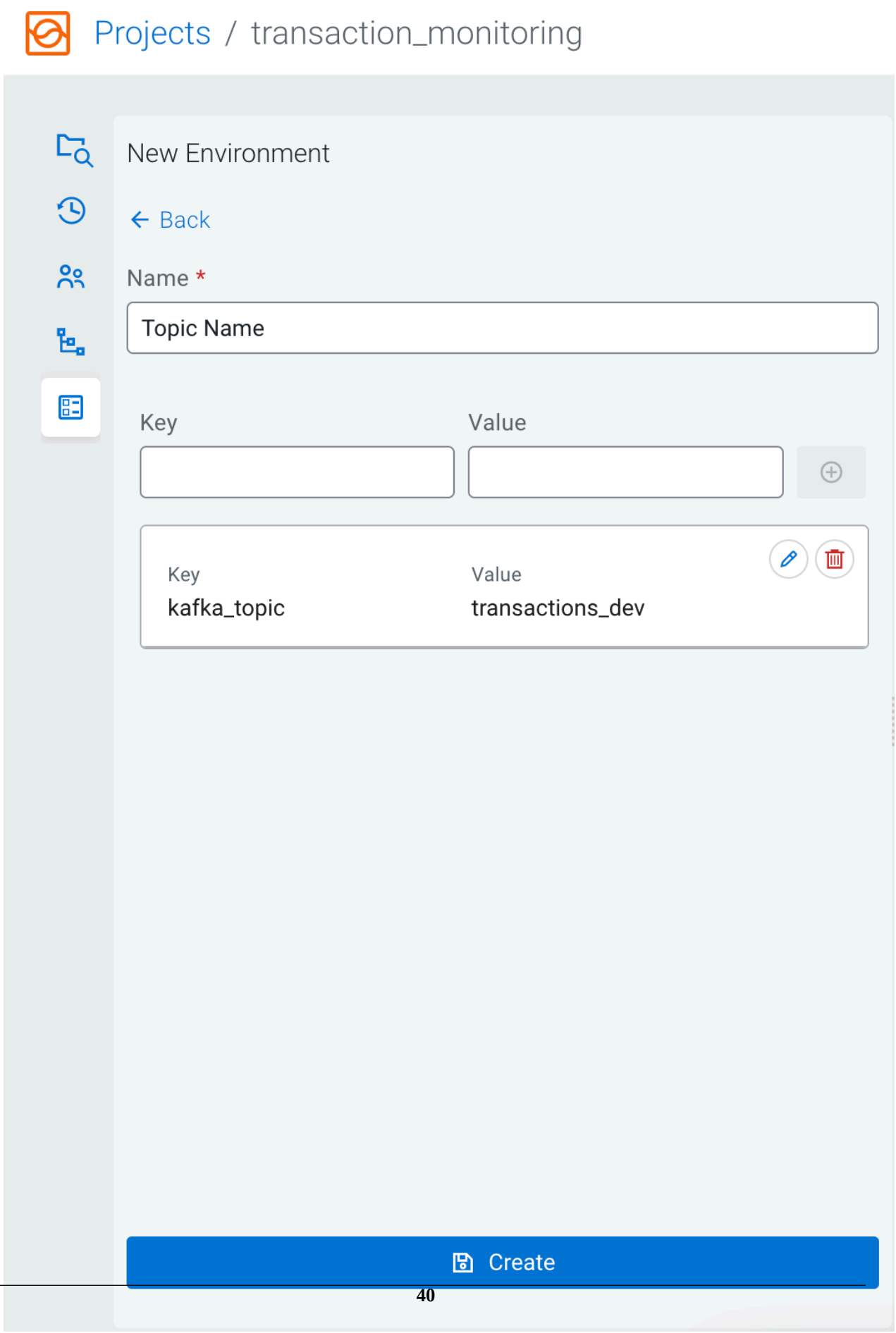
- Properties of tables in DDL, for example topic names, connector types and so on

- Kafka data source and catalog properties, for example brokers, schema registry url and so on

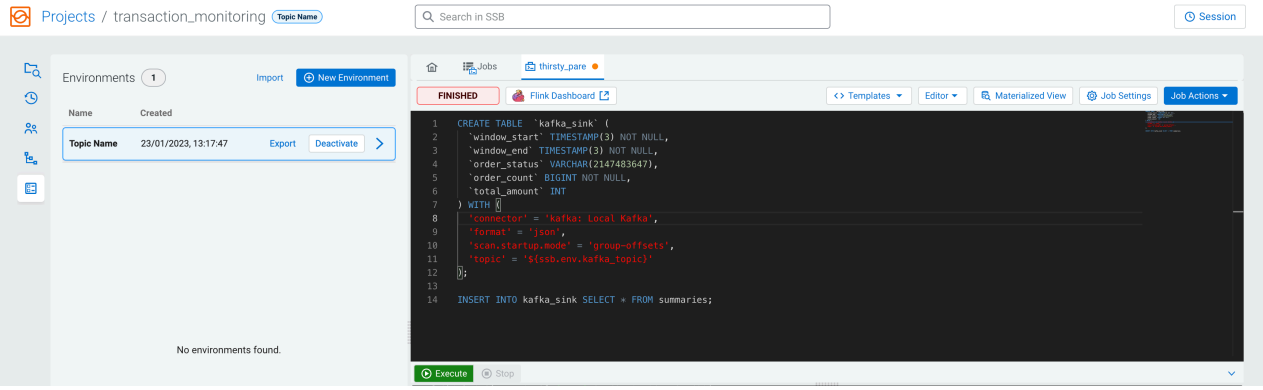
This helps you import and export your project between environments and users. For example, you can create a variable for the Kafka topic name. For example, you could use a placeholder to distinguish between the Kafka topic in a development or production environment. This means that instead of specifying the Kafka topic name in the CREATE TABLE statement as my_topic, you would use a placeholder based on the key and value provided in the environment file as shown below. When the project is synchronized to Git, and imported to a production environment, the topic name will be substituted with the applicable Kafka topic name, if an Environment is also configured on that SSB instance. You can also export and import Environments as JSON files, which make the management of environments easier for users.

To create an Environment configuration, select the **Environments** view and click New Environment.

Provide a name to the Environment, and add a key with dedicated value, and click on the plus button.


Projects / transaction_monitoring
New Environment
Back
Name *
Topic Name
Key
Value
Key
kafka_topic
Value
transactions_dev
Create

Click on the Save button, and you are redirected to the **Environments** view. Click on Activate next to the Environment you would like to use to the currently working on project.



Now that you have an Environment, you can use the variable (prefixed with ssb.env.) in the CREATE TABLE statement as shown in the following example:

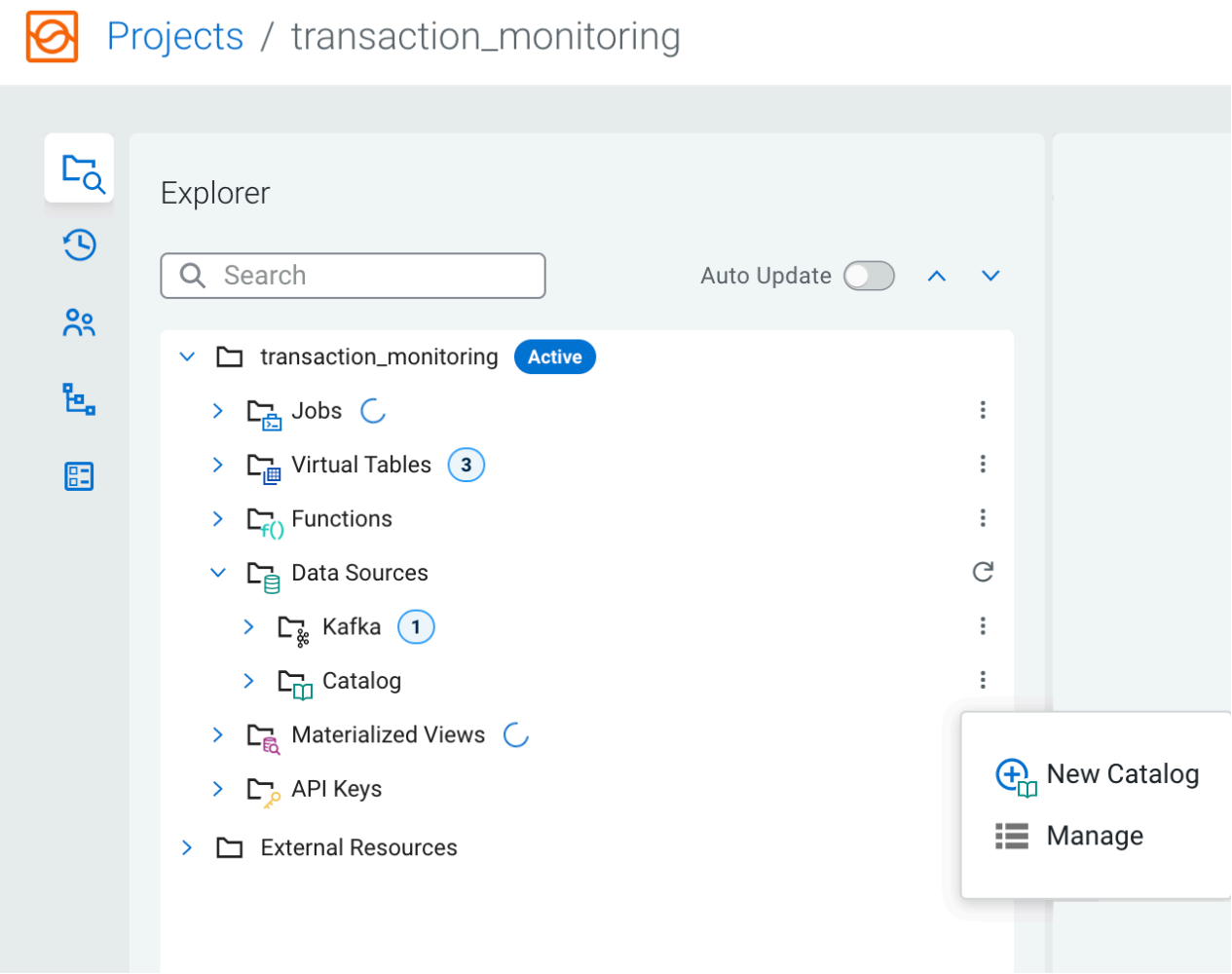
```
CREATE TABLE  `kafka_sink` (
  `window_start` TIMESTAMP(3) NOT NULL,
  `window_end` TIMESTAMP(3) NOT NULL,
  `order_status` VARCHAR(2147483647),
  `order_count` BIGINT NOT NULL,
  `total_amount` INT
) WITH (
'connector' = 'kafka: Local Kafka',
'format' = 'json',
'scan.startup.mode' = 'group-offsets',
'topic' = '${ssb.env.kafka_topic}'
);
```

Adding a sink using a Schema Registry catalog

You can use Schema Registry as a catalog to read data or output results to a Kafka topic based on a defined schema in Schema Registry.

SQL Stream Builder (SSB) in the CSP Community Edition comes with a preconfigured Schema Registry catalog, which is available by default in the ssb_default project. This means that if you are working in the ssb_default project, you do not have to register the local Schema Registry service running in the Community Edition as it is already set up.

If you have created a new project, you need to register the Schema Registry catalog using the **Catalog** page that can be accessed from the **Explorer** by expanding the Data Sources, and clicking ⋮ New Catalog .

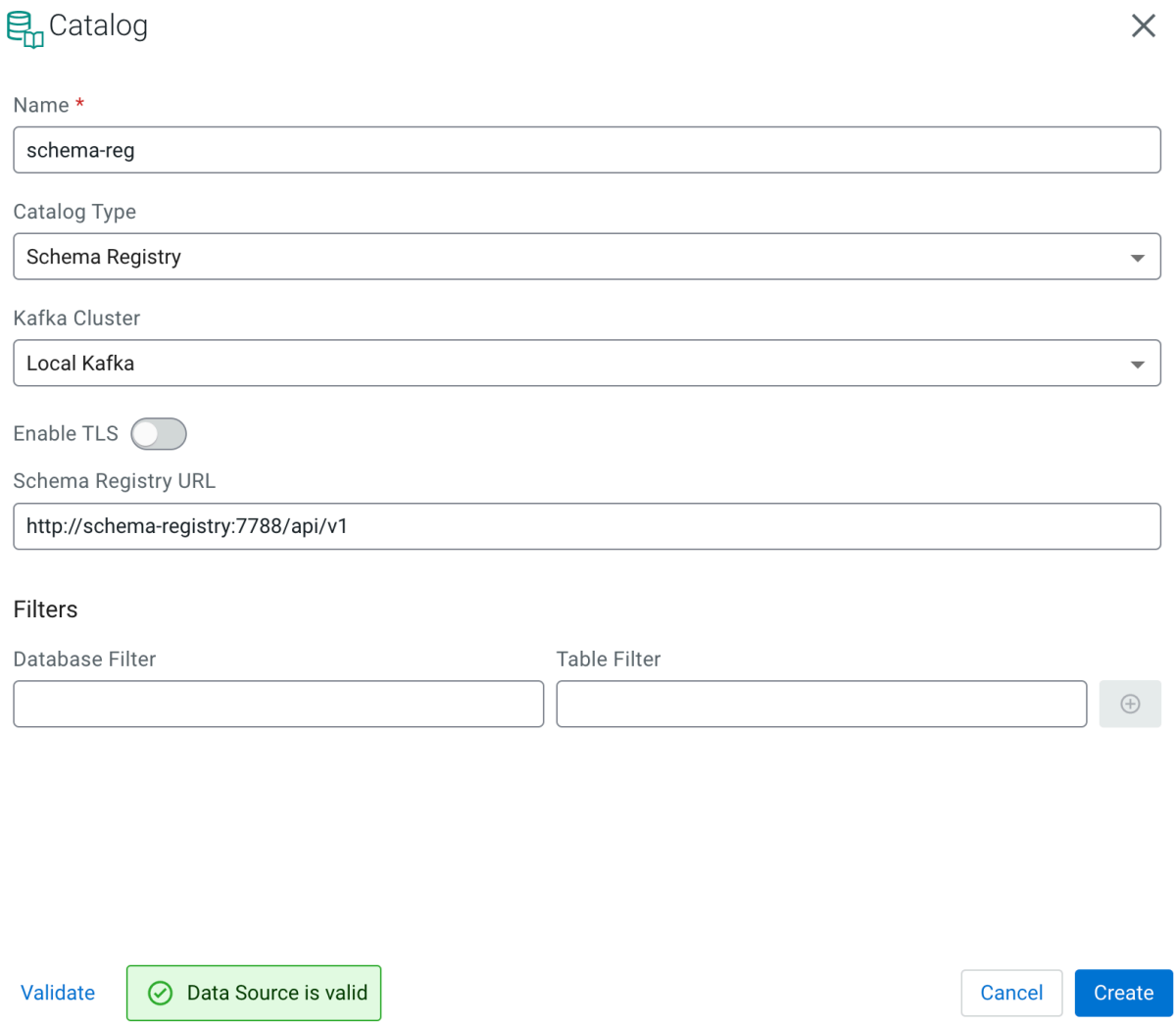


Provide the following information to register the Schema Registry catalog:

Setting	Value
Name	schema-reg
Catalog Type	Schema Registry
Kafka Cluster	Local Kafka
Schema Registry URL	http://schema-registry:7788/api/v1

After providing every needed information, click on the Validation button to verify the catalog. If it is successful, click on the Create button to add Schema Registry as a catalog.

The following illustration also details how you need to fill out the **Catalog** page:

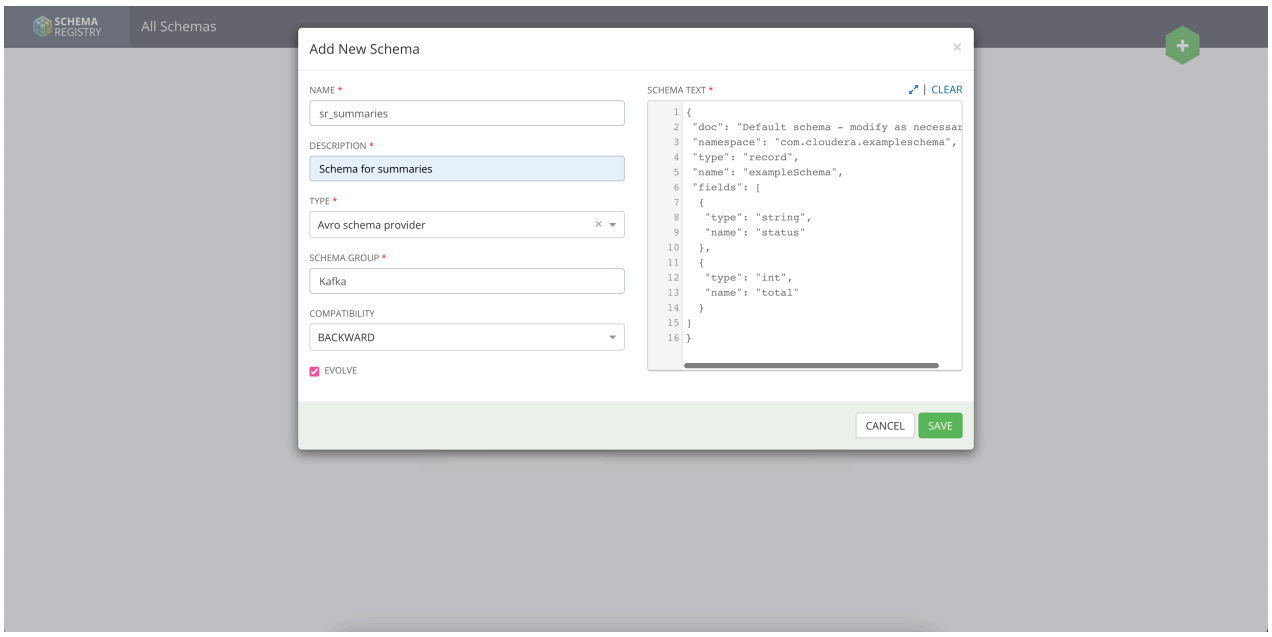


After adding Schema Registry as a catalog, you need to create a schema and a topic that will be used as the output sink.

Navigate to the Schema Registry user interface by opening http://localhost:7788 from your browser. Click on the Add New Schema button, and provide the schema and the necessary information to the window:

Setting	Value
NAME	sr_summaries
DESCRIPTION	Provide a description based on your requirements.
TYPE	Avro schema provider
SCHEMA GROUP	Kafka
COMPATIBILITY	BACKWARD

The following illustration also details how you need to fill out the **Add New Schema** settings.



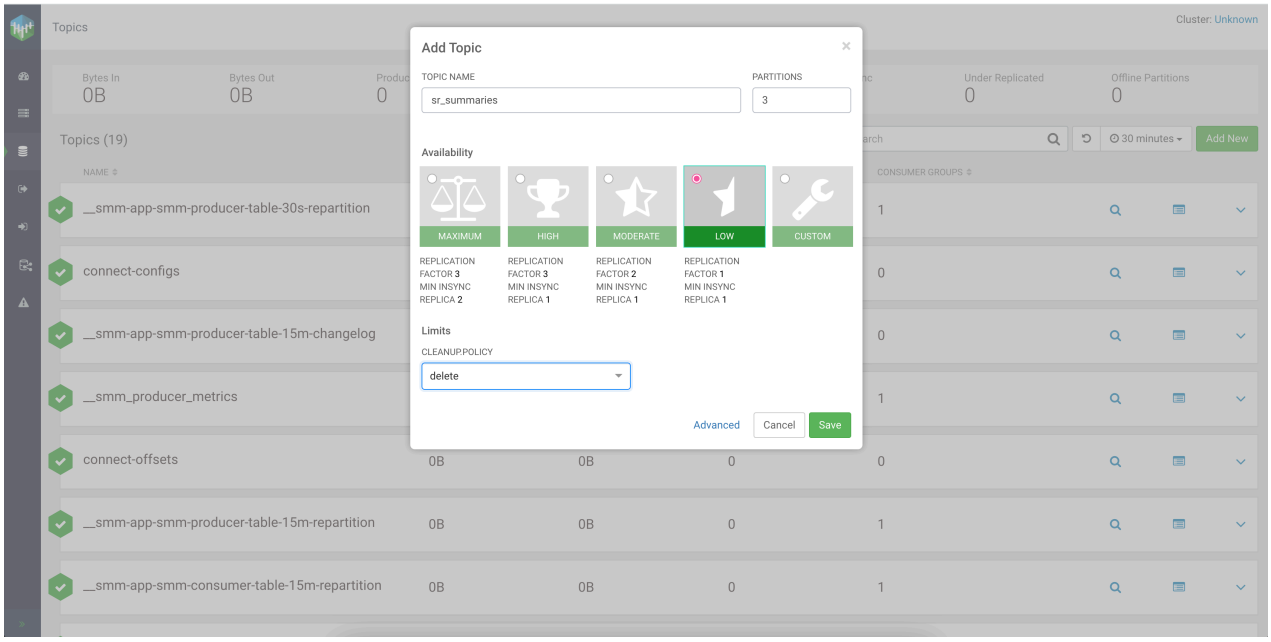
The following schema was used in the example:

```
{
 "doc": "Default schema - modify as necessary",
 "namespace": "com.cloudera.exampleschema",
 "type": "record",
 "name": "exampleSchema",
 "fields": [
  {
   "type": "string",
   "name": "status"
  },
  {
   "type": "int",
   "name": "total"
  }
]
}
```

After you created the schema, you need to create a new Kafka topic with the same name. You can easily create a Kafka topic using Streams Messaging Manager (SMM). Navigate to the SMM user interface by opening http://localhost:9991 from your browser. Select Topics from the main menu, and click on the Add New button. Fill out the fields, and click Save to create the topic:

Setting	Value
TOPIC NAME	sr_summaries
PARTITIONS	3
Availability	LOW
CLEANUP POLICY	delete

The following illustration also details how you need to fill out the **Add Topic** settings.

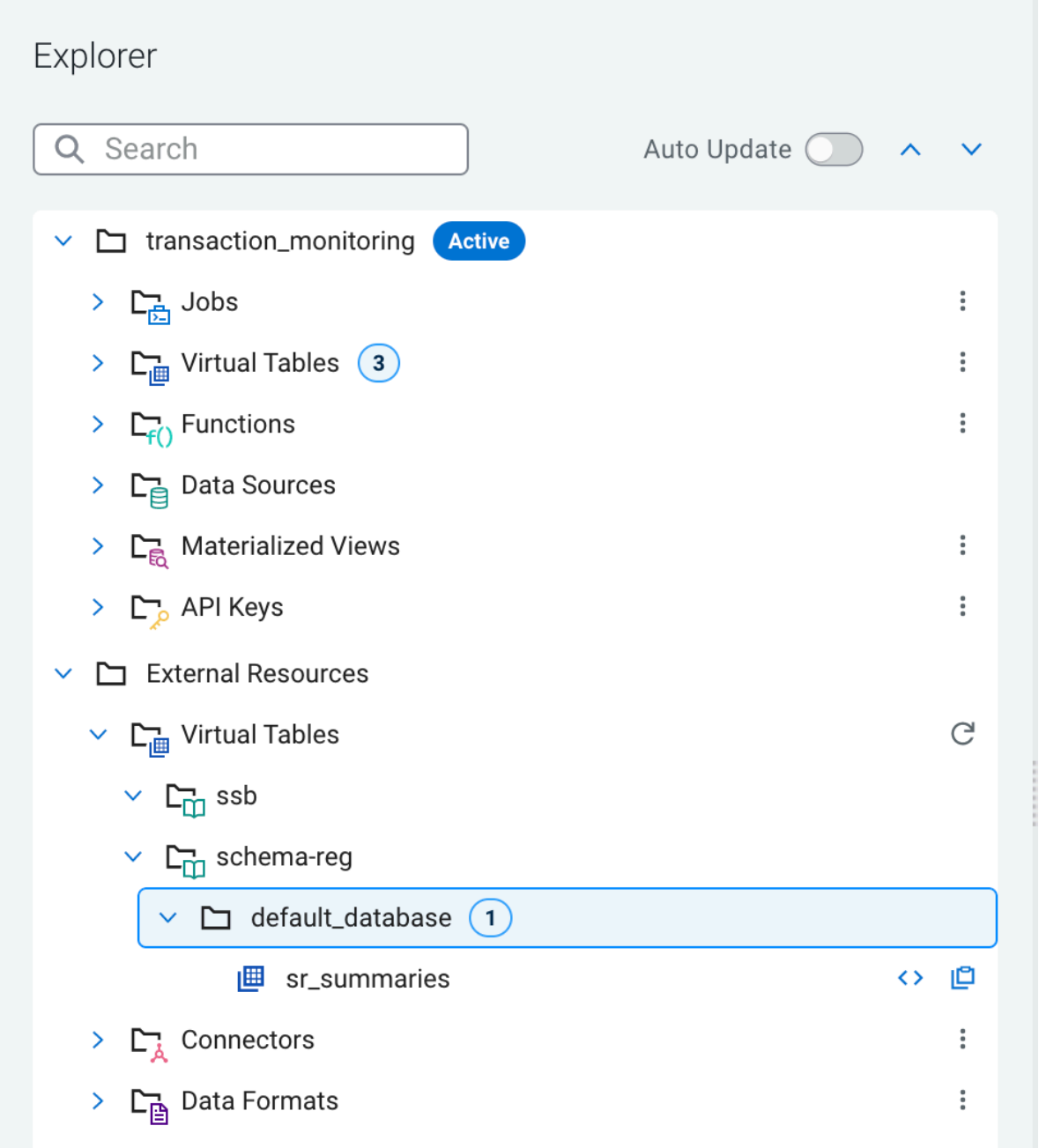


Important: Make sure that the name of the topic matches the name of the schema you have previously created.

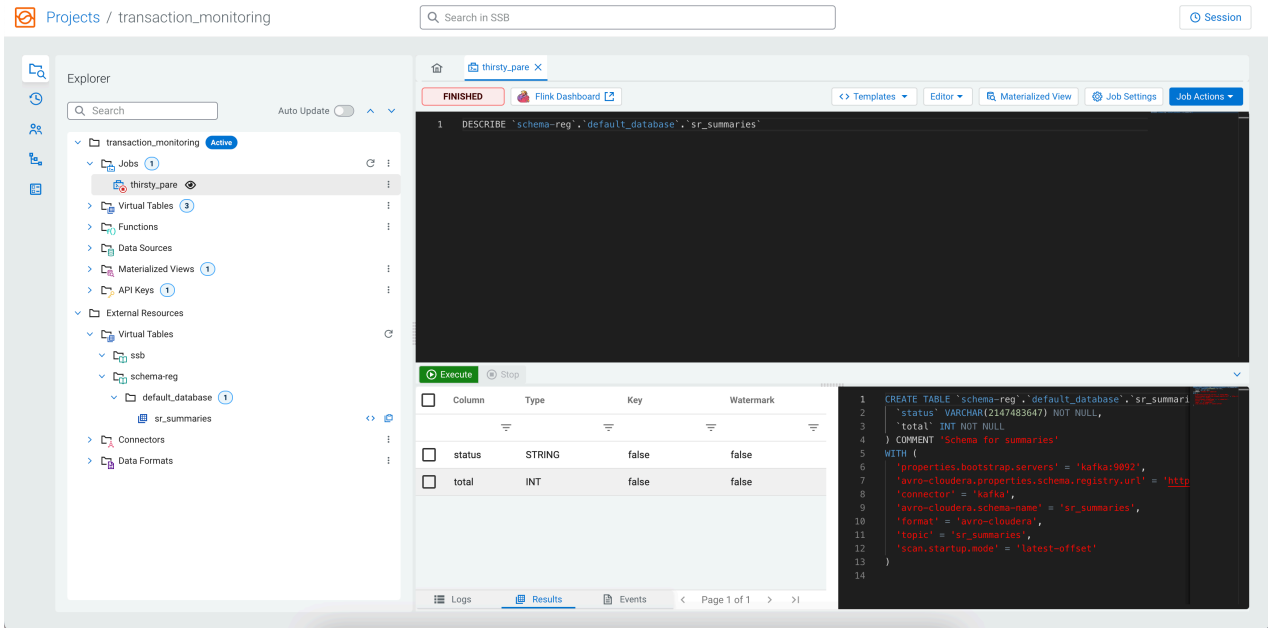
Navigate back to the Streaming SQL Console using http://localhost:18121 or refresh the browser window, and the table will appear under **External Resources > Virtual Tables**.

Note:

To view the tables created from catalogs, you need to expand the External Resources directory and then drill down to the Schema Registry catalog.



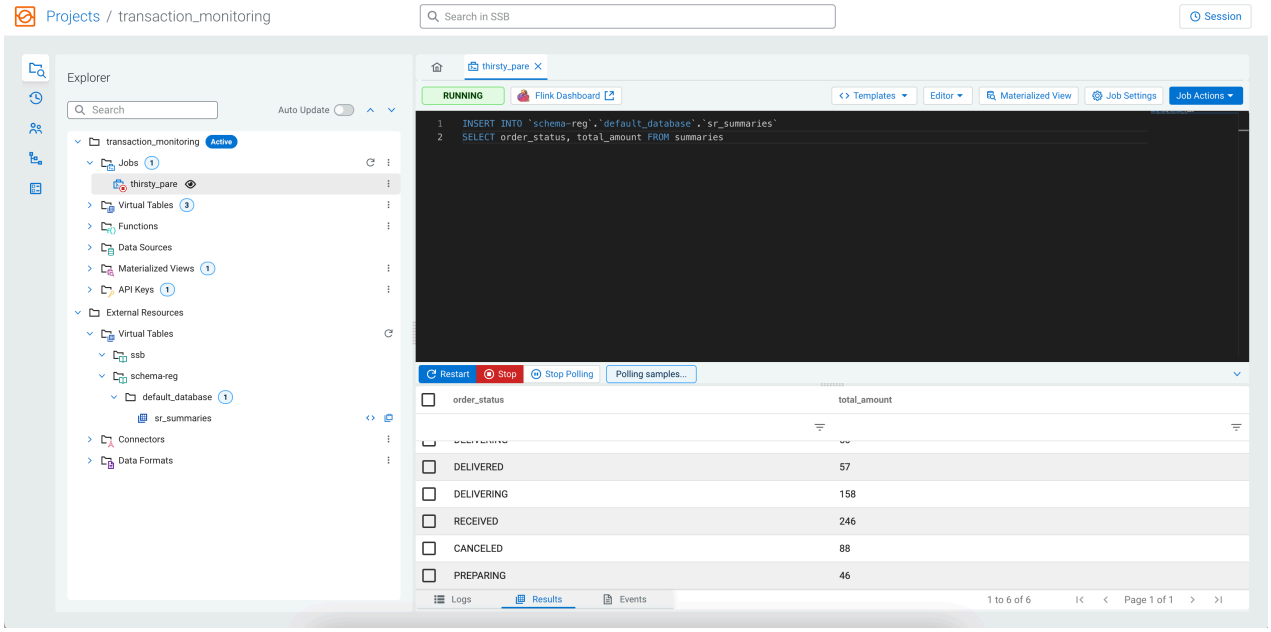
When you describe the created table, you can see the added schema and also that it is automatically connected to the local Kafka and Kafka topic.



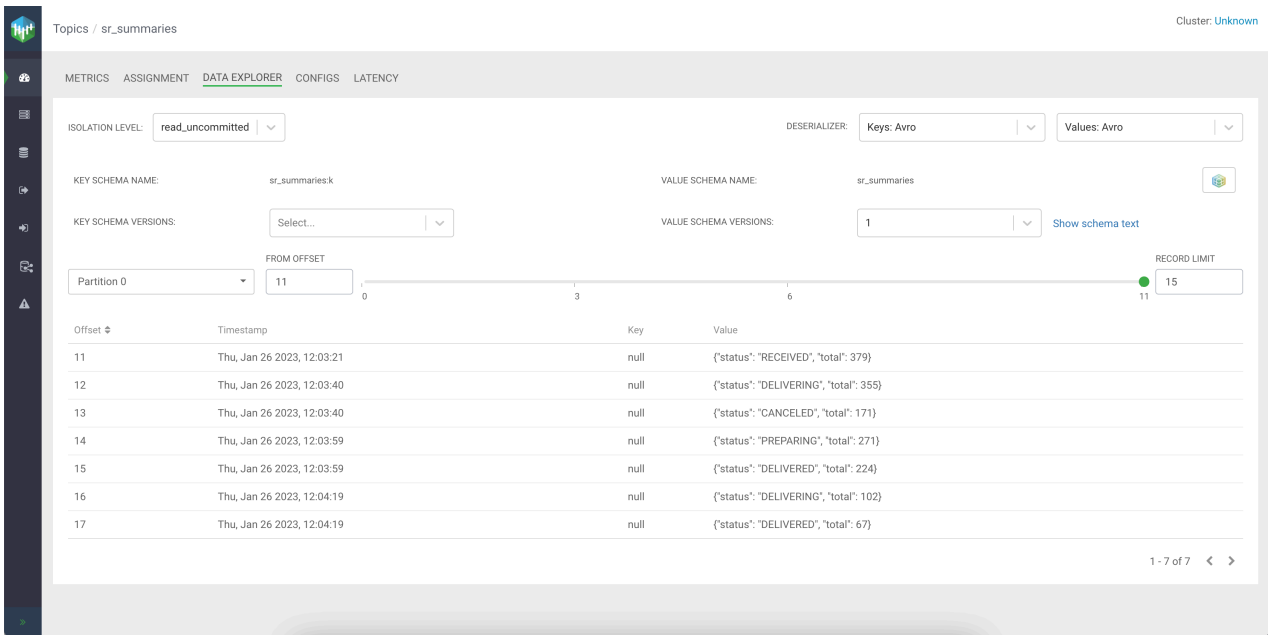
After setting up the schema and topic, you can submit a SELECT query where the order_status and total_amount are inserted into the sr_summaries table.

```
INSERT INTO `schema-reg`.`default_database`.`sr_summaries`
SELECT order_status, total_amount FROM summaries
```

A sample of the results is displayed on the browser under the **Results** tab, however you can also view the messages generated to the Kafka topic using SMM.



When you navigate to the SMM user interface, search for the sr_summaries topic, and click on the Data Explorer view to see the received messages. Set the **Values deserializer type** to Avro for the messages to be shown correctly. SMM will automatically fetch the schema from Schema Registry to deserialize the messages.



Adding a Materialized View

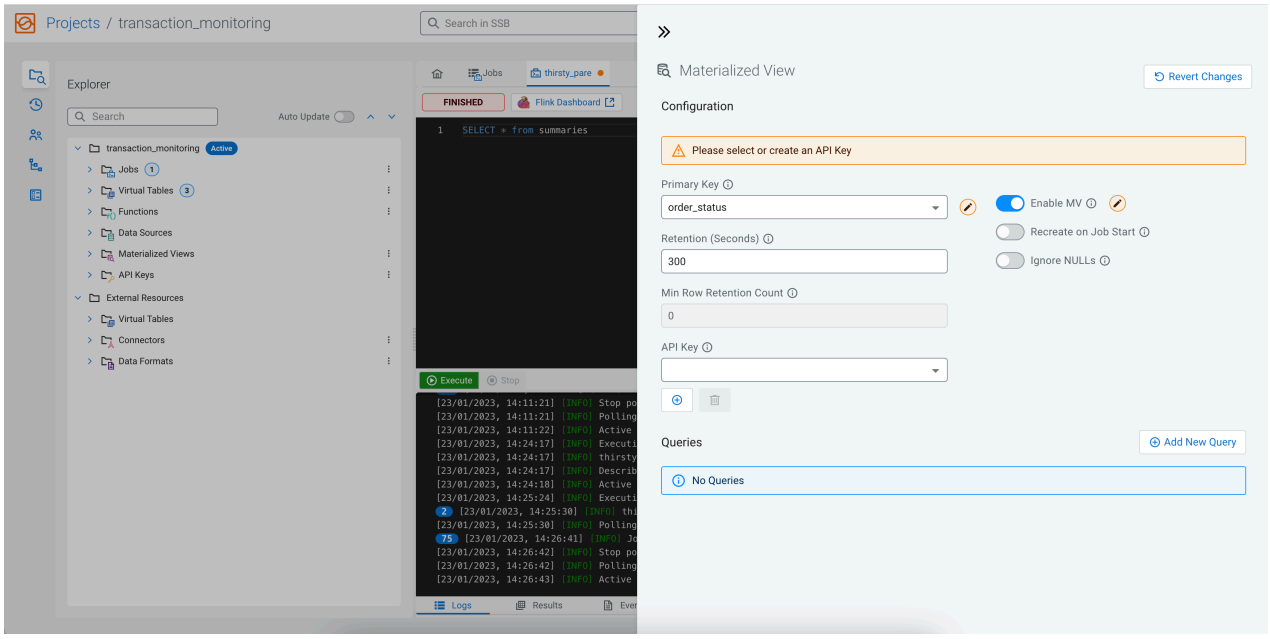
One of the features of SQL Stream Builder is creating Materialized Views. Materialized Views let you create a persistent snapshot of the stream that can be queried through a REST endpoint. This can then be queried by external applications and tools. Now that you know how to query simple and aggregated data, learn how to create Materialized Views.

In the following example, you will create a Materialized View that shows the latest order count and total amount for each order status. The key will be the status, and the other fields will be updated with the value from the latest 20 second interval.

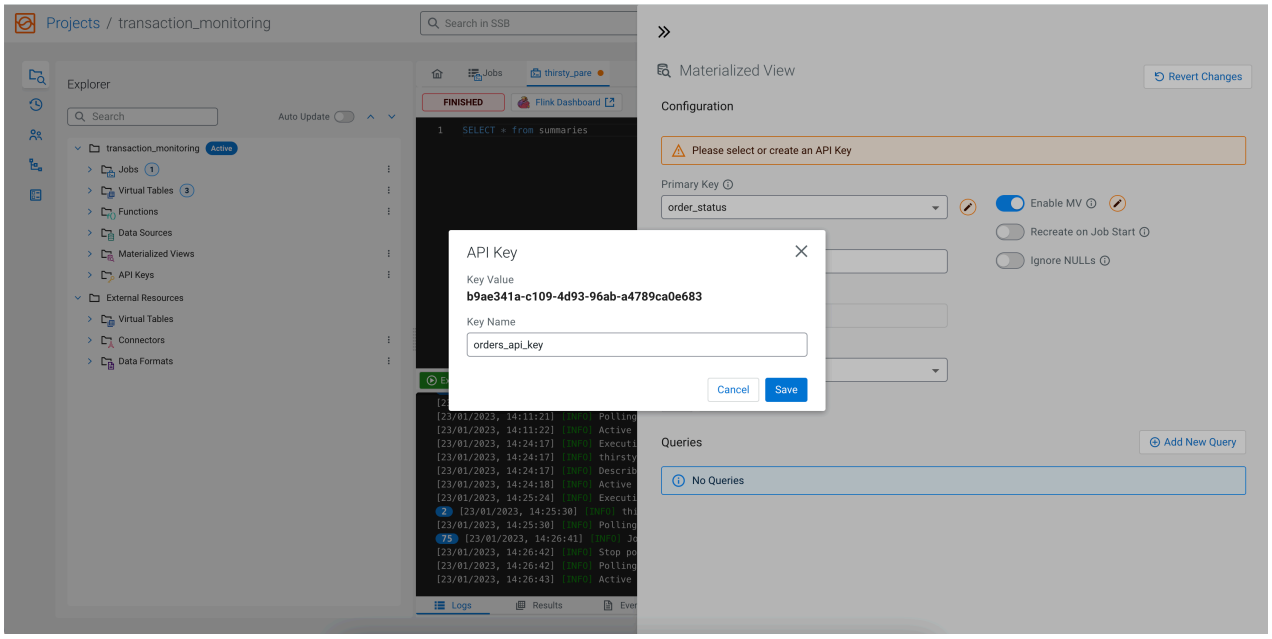
First, you need a New Job with a SELECT query. Add the following SQL query to the SQL Editor to run the query:

```
SELECT * FROM summaries
```

Before executing the query, you need to enable the Materialized View feature. Click on the Materialized View button, set the Materialized View configuration to *Enabled*, and select the *order_status* as the primary key.

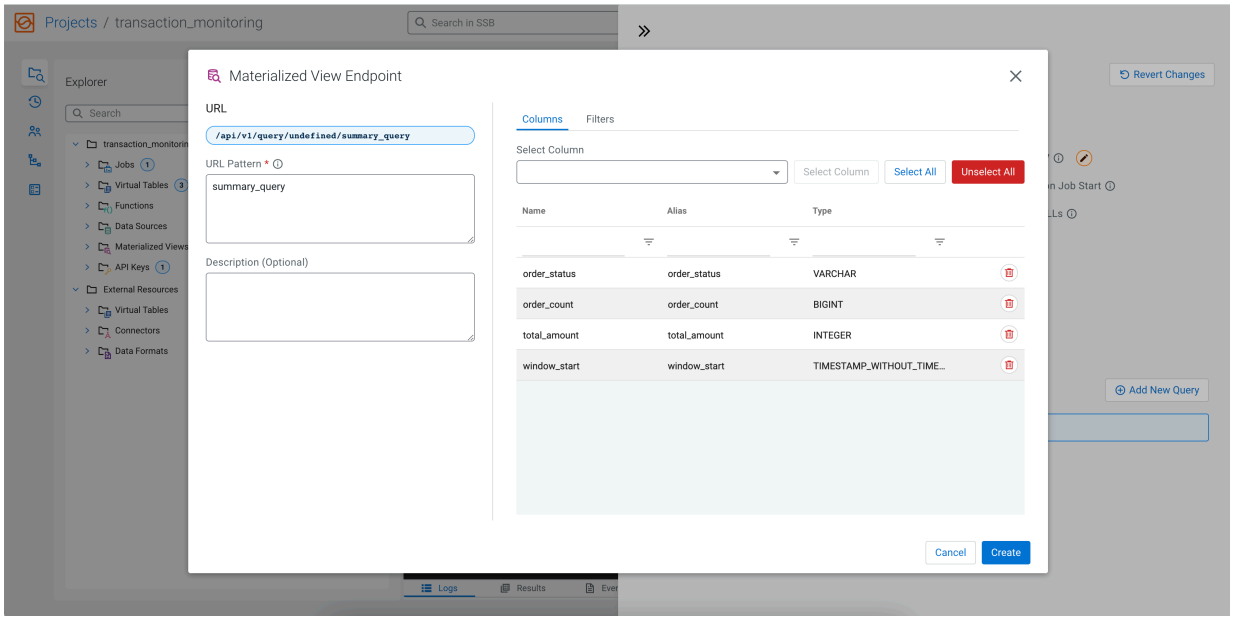


You also must add an API key to the configuration that will be a part of the generated URL. Click on the Add API Key button, and provide any type of name to the API key. For example orders_api_key, and click Save. After the API key is created, select it as the API Key of the current Materialized View.

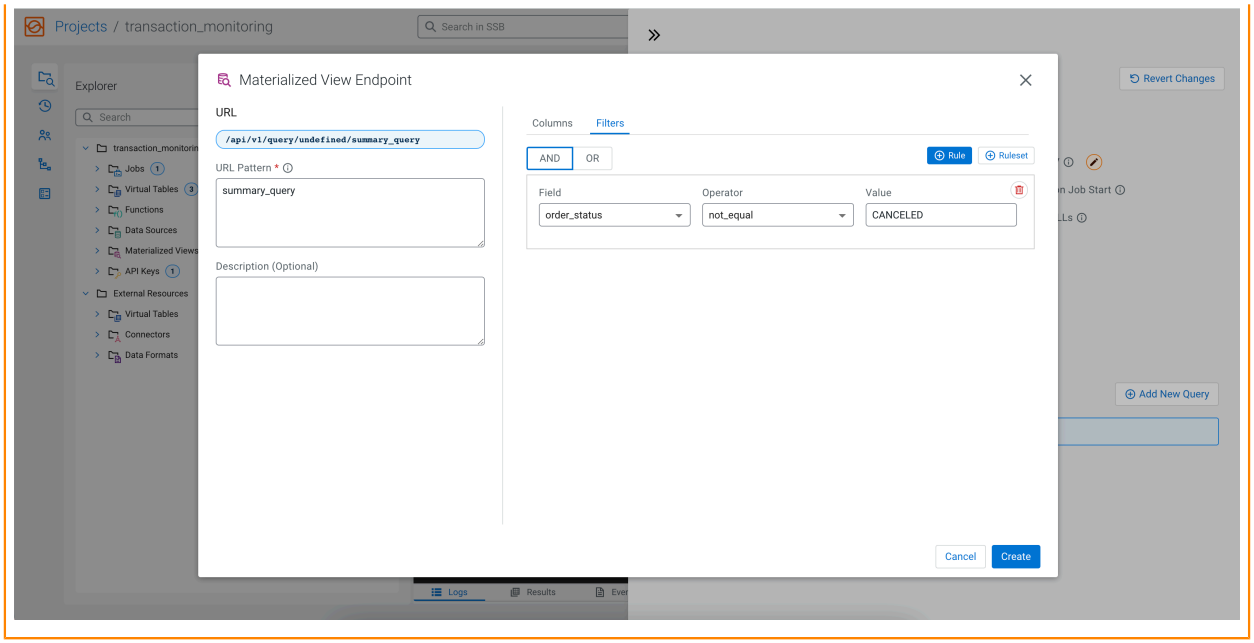


After configuring the Materialized View, you can add a query by clicking on the Add New Query button. On the **Query Configuration** page, specify the URL Pattern and select the fields that you want to include in the Materialized View. You can also add filters to further narrow down the result. In the following example, summary_query is the URL Pattern, order_status, order_count, total_amount and window_start columns are selected as fields. You can add these fields by clicking Select All, and removing window_end with the red bin icon under **Actions**. You also have the option to select the fields from the drop-down menu under **Query Builder**, and add the fields individually. A filter is also added to exclude the orders with a *CANCELED* status.

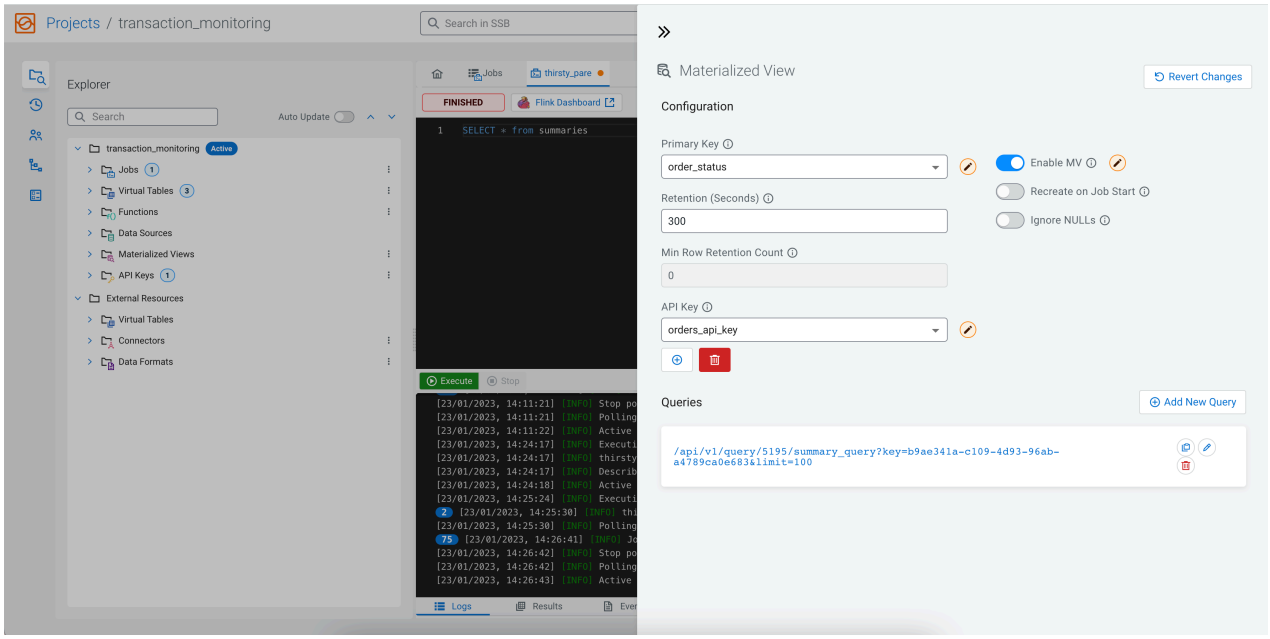
For Columns tab



For Filters tab



Click on the Create button to add the query to the Materialized View. The URL will appear under the Materialized View Queries.



When you click on the URL, you are redirected to the http://localhost:18131 address. You will receive an error message that no data is generated, because the SELECT query is not executed yet. Go back to the SQL Editor using the arrows on the **Materialized View** window, and click on the Execute button to start the SQL job. After a couple of seconds, the endpoint becomes active and the results show up on the localhost address.

localhost:18131/api/v1/query/5198/summary_query?key=ae4b5d47-04e0-4ca1-aebe-d069daee7b46


```
[
  - {
      order_status: "DELIVERING",
      order_count: "4",
      total_amount: "255",
      window_start: "2021-10-25 16:14:20.0"
    },
  - {
      order_status: "PREPARING",
      order_count: "5",
      total_amount: "197",
      window_start: "2021-10-25 16:14:20.0"
    },
  - {
      order_status: "RECEIVED",
      order_count: "3",
      total_amount: "177",
      window_start: "2021-10-25 16:14:20.0"
    },
  - {
      order_status: "DELIVERED",
      order_count: "3",
      total_amount: "199",
      window_start: "2021-10-25 16:14:20.0"
    }
]
```


Next step

You have completed the Getting Started guide of Streaming Analytics. As you already know the basic steps to use SSB, feel free to experiment more and get more familiar with the features of SSB by trying out the PostgreSQL CDC and Filesystem connectors, or learning how to monitor your SQL jobs.

Using different connectors in SSB

You can use the supported connectors to create end-to-end streaming analytics processes based on your requirement.

Using Filesystem source

You can use the Filesystem connector in the Community Edition to process files which are stored on the filesystem.

To use the Filesystem connector, you need to either copy the files to the persistent docker volume provided with the Community Edition, or map bind mounts from the host filesystem.

For Copy files to Docker volume

How to copy files to the Docker volume?

To copy existing files directly to the cluster for processing, you can use the docker command to copy them directly to the persistent volume.

First, determine the name of the Job Manager container. You can do this by running the following command:

```
$ docker ps -f "label=com.docker.compose.service=flink-jobmanage
r" --format="{{.Names}}"
csp-ce_flink-jobmanager-1
```

In the examples below, csp-ce_flink-jobmanager-1 is the name of the container.

Next, copy your file to the shared volume in the container. For example, if you have a file in your local directory called example.json that you would like to process, you can copy it to the cluster with the following docker command:

```
docker cp example.json csp-ce_flink-jobmanager-1:/persistent
```

Make sure to substitute the container name for the one you located in the previous step.

To check that the file is successfully copied, first connect to the container with the following command:

```
docker exec -it csp-ce_flink-jobmanager-1 bin/bash
```

When you access the container, go to the persistent directory and list out the files there:

```
cd /persistent
ls
```

The example.json should be listed in the persistent directory.

When you create a table using the Filesystem connector as shown in *Creating a Filesystem backed table*, you need to reference the path of the copied file in the container: file:///persistent/example.j son

For Bind mounting in Docker

How to make files accessible using bind mount in Docker?

If you have many or large files, it is easier to reference a directory on the host machine using a bind mount as you do not need to copy the files to the volumes. To make the directories available, open the docker-compose.yml file in an editor, and locate the entry for the Flink Job Manager, and add the references to your files:

```
flink-jobmanager:
        …
    volumes:
      - flink-volume:/persistent
      - /your/local/volume:/mnt/local0
      - /your/other/volume:/mnt/local1
```

Once you have added the volumes to the flink-jobmanager entry, you need to add them to the flink-taskmanager entry as well.

When you create a table using the Filesystem connector as shown in *Creating a Filesystem backed table*, you need to reference the path of the mounted file in the volume:

file:///mnt/local0/example.json

Creating a Filesystem backed table in Streaming SQL Console

After setting up the local file, open the Streaming SQL Console, and execute the following example:

```
DROP TABLE IF EXISTS my_file_source;
CREATE TABLE `ssb`.`ssb_default`.`my_file_source` (
  `foo` VARCHAR(2147483647)
) WITH (
  'format' = 'json',
  'path' = 'file:///persistent/example.json',
  'connector' = 'filesystem'
);
```

```
SELECT * FROM my_file_source;
```

This example parses JSON serialized input with rows delimited by a newline. When using JSON, the columns defined match the keys in the JSON document.

The Filesystem connector also supports a number of other formats, such as CSV or raw. The raw format is a good option for arbitrary format log files with no specific serialization. In this case, each line of data (delimited by newline) is returned in a single column, and can be split or parsed using User Defined Functions, SQL string functions, regular expressions, and so on.

Using PostgreSQL CDC connector

The Community Edition comes pre-configured with a PostgreSQL database that can be used to try Change Data Capture (CDC). After setting up a table in the PostgreSQL database, you can use the connector for your SQL queries.

A database named ssb_cdc has been created with the necessary configuration. An ssb_cdc role (password: cloudera) has also been added that can be used to access the changelog streams.

Before using the PostgreSQL connector, you need to up a CDC table in the PostgreSQL database.

After accessing the PostgreSQL instance, start the psql client and connect to the ssb_cdc database:

```
docker exec -it <psql_container_name_or_id> /bin/bash
psql
\c ssb_cdc
```

As an example, create the following table in PostgreSQL:

```
CREATE TABLE cdc_table (a varchar, b int);
```

Set the permissions to the created table:

```
ALTER TABLE cdc_table REPLICA IDENTITY FULL;
```

After setting up the database, you can use it in Streaming SQL Console.

To subscribe to the changelog stream in SSB, create a table and check the data in the database using the following example:

```
CREATE TABLE `ssb`.`ssb_default`.`cdc_postgres_table` (
  `a` VARCHAR(2147483647),
  `b` INT,
  `ts` AS PROCTIME()
) WITH (
  'connector' = 'postgres-cdc',
  'hostname' = 'postgresql',
  'database-name' = 'ssb_cdc',
  'schema-name' = 'public',
  'table-name' = 'cdc_table',
  'username' = 'ssb_cdc',
  'password' = 'cloudera',
  'decoding.plugin.name' = 'pgoutput',
  'debezium.slot.name' = 'flink'
);

SELECT * from cdc_postgres_table;
```

Insert or update a row in the PostgreSQL table using your database tool of choice.

For example, you can use the psql client and manually insert values to the cdc_table:

```
psql
\c ssb_cdc
```

```
insert into cdc_table values ('Hello', 123);")
```

A row representing the change will show up in the **Results** tab.

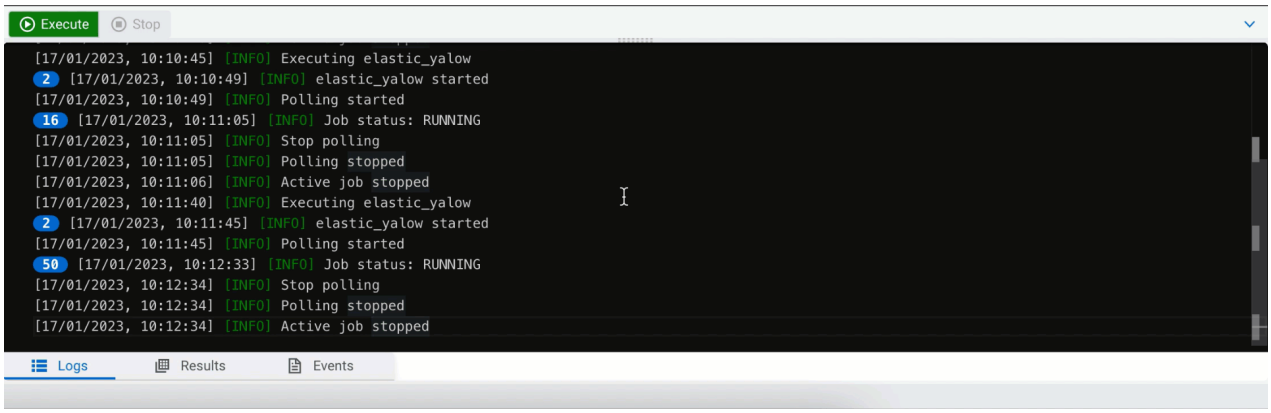
Monitoring SQL results and jobs

You can check your SQL query results and monitor your SQL jobs with Streaming SQL Console, Flink Dashboard and Streams Messaging Manager.

Sampling results in Streaming SQL Console

You can sample data from a running job to the Streaming SQL Console without writing it to any sink to inspect the data and to make sure the job is running correctly, and producing the results you expect.

Sampling the results to your browser allows you to inspect the queried data and iterate on your query. You can sample 100 rows in the **Results** tab under the **SQL Editor** on the jobs page. Even if you do not specify any sink to the SQL job, the results automatically appear in the **Results** tab.

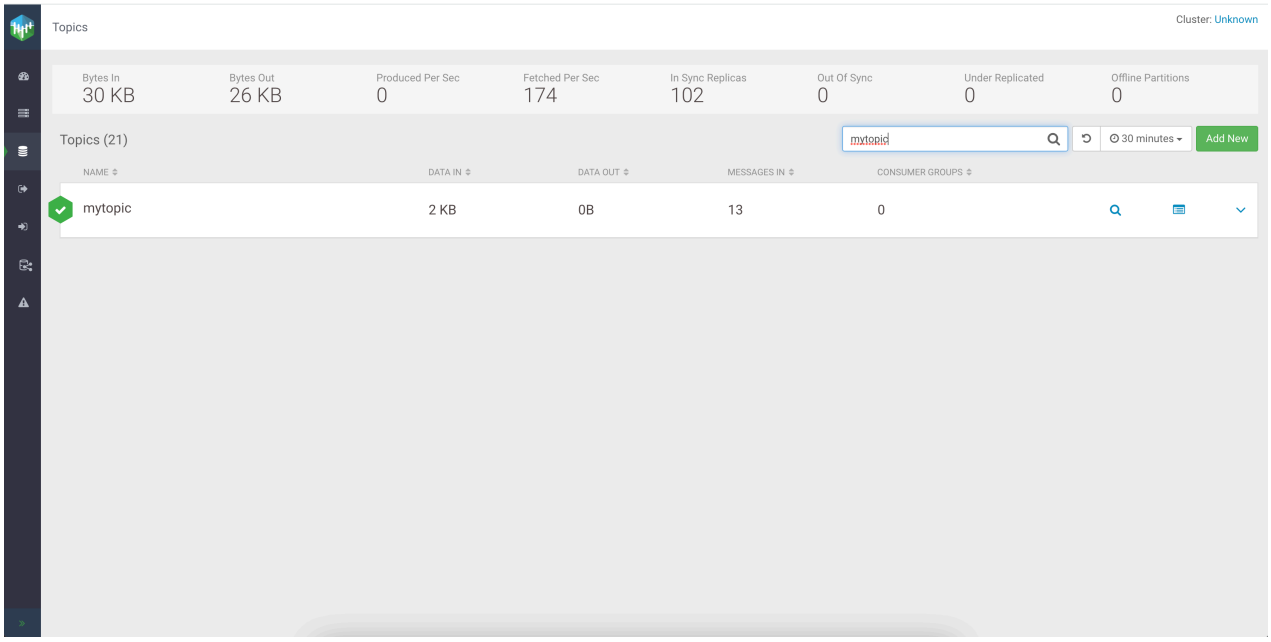


Reviewing data in Streams Messaging Manager

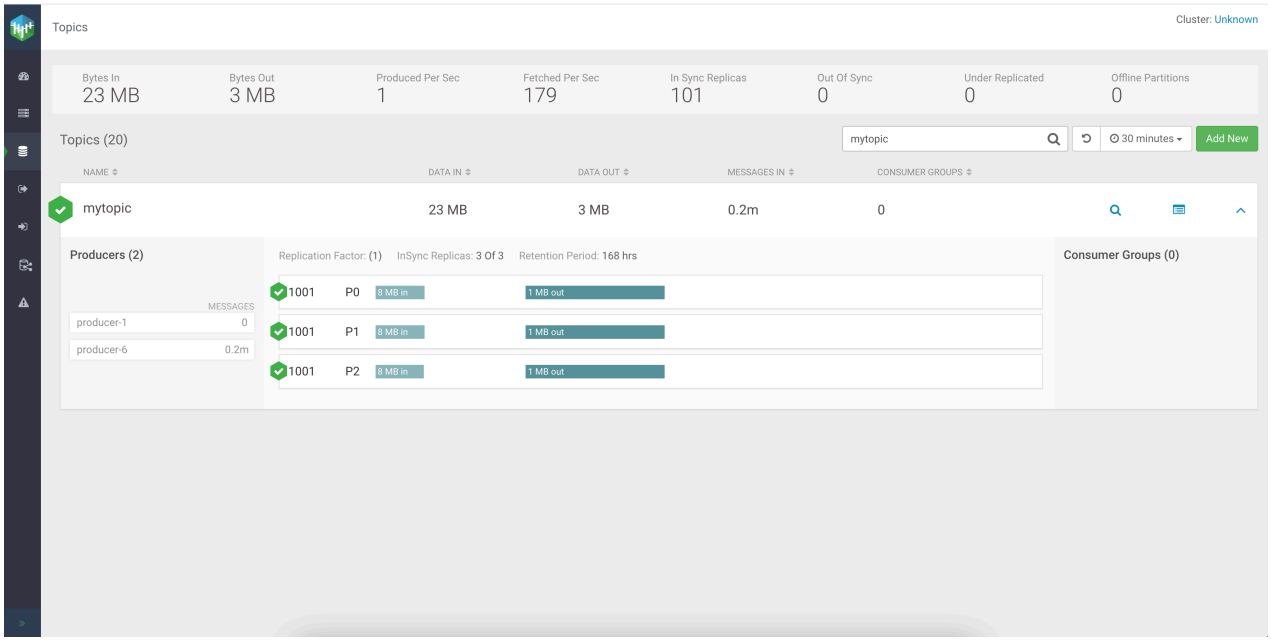
When you create a job that sends results to a Kafka sink, you can review if the results are being written out to the Kafka topic using Streams Messaging Manager (SMM).

Without stopping the previously started job that writes results from the summaries table, access the Streams Messaging Manager (SMM) user interface from your browser: http://localhost:9991/#/topics

On the SMM user interface, navigate to Topics and search for the previously created Kafka topic: mytopic.



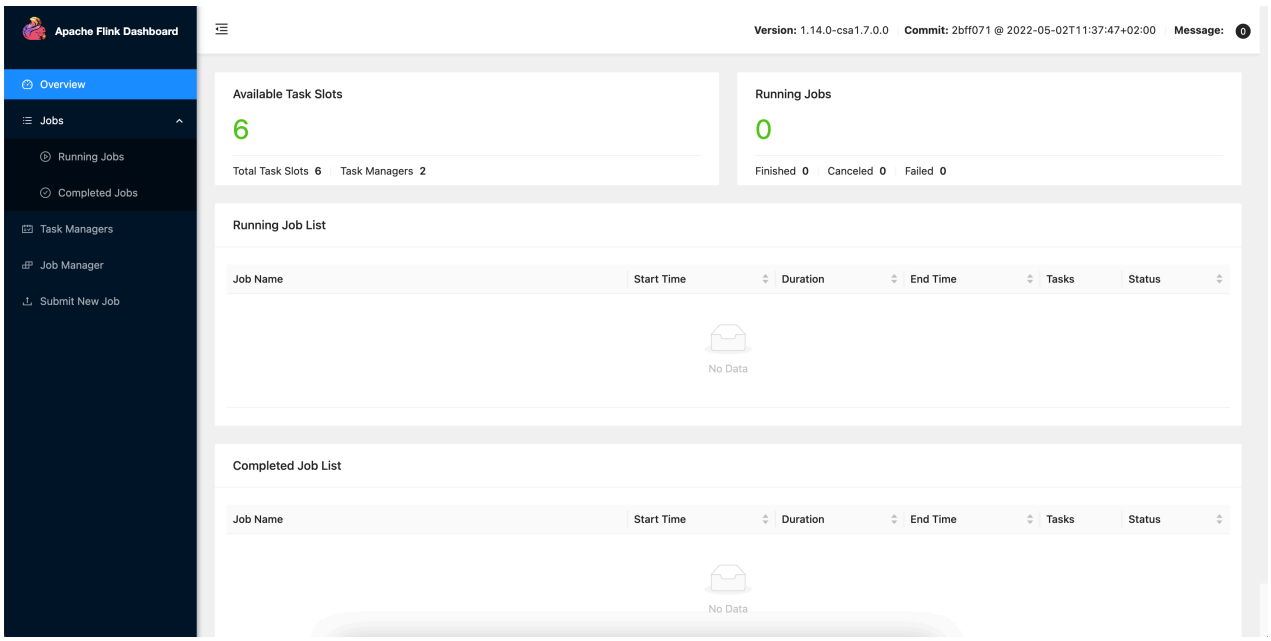
When the mytopic shows up, you can already see that there are messages already in the topic. However, you can further look into the topic by clicking on the entry to see more details about the producers and messages.



Reviewing jobs in Flink Dashboard

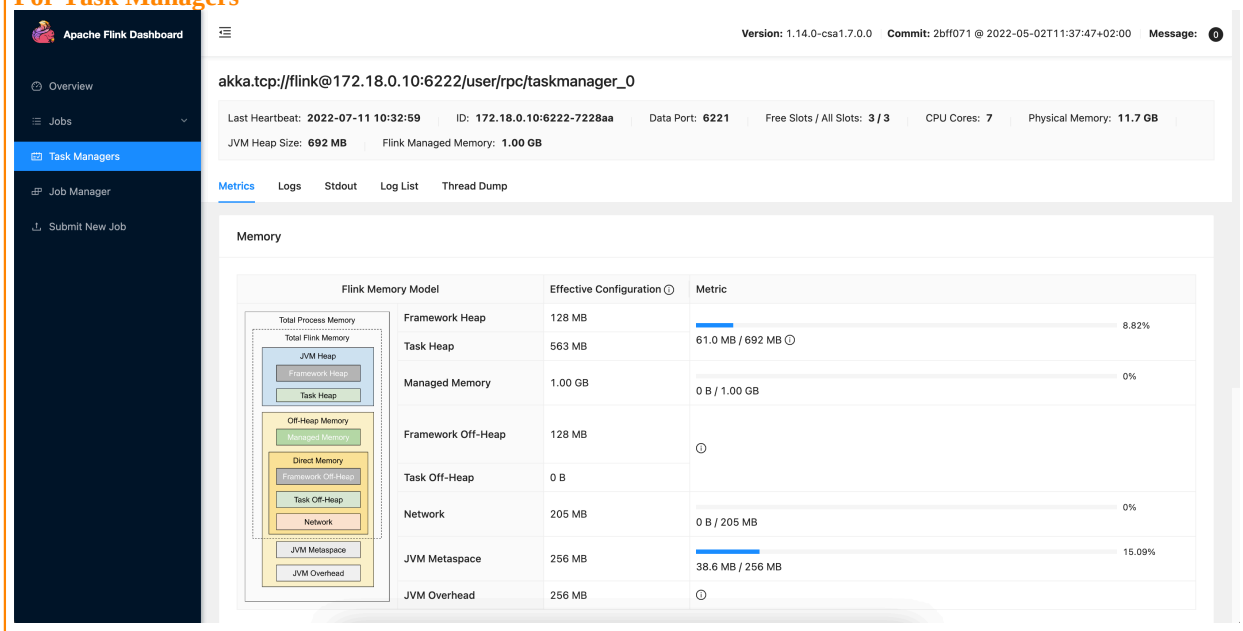
As Flink is the engine running under SQL Stream Builder, you can use the Flink Dashboard to review your Flink jobs running as SQL jobs on the Streaming SQL Cosole.

You can reach the Flink Dashboard from Streaming SQL Console main menu, from the SQL Editor or you can access it using http://localhost:8081.

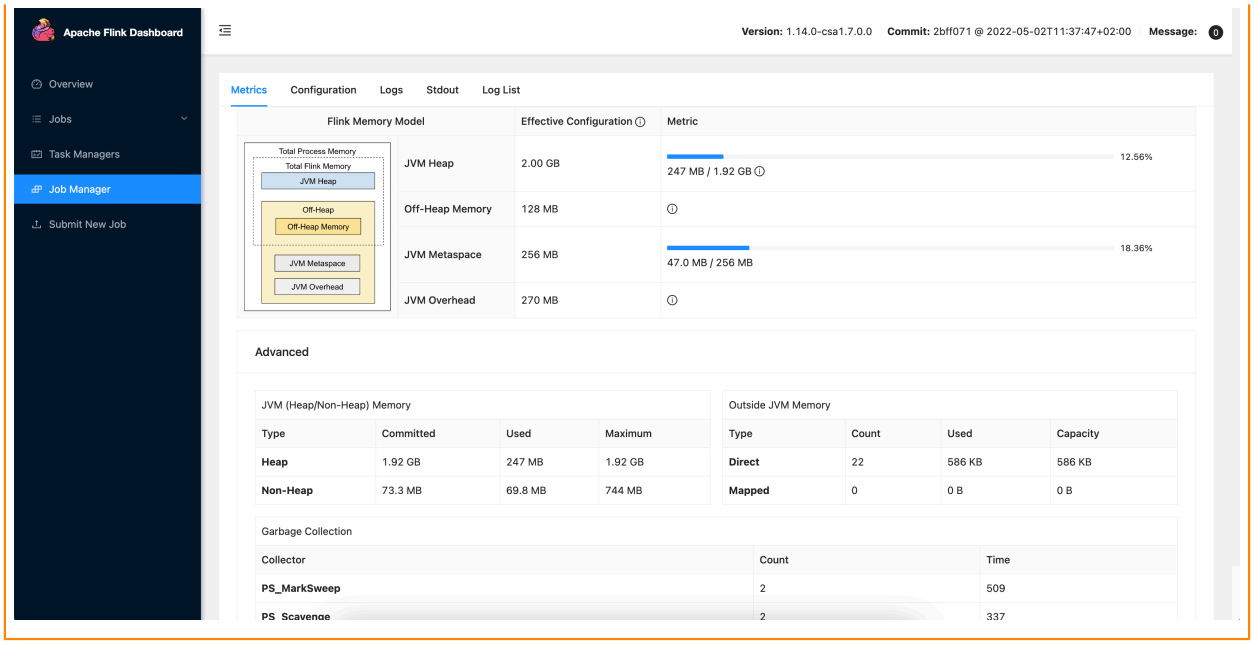


Other than checking the running and completed jobs, you can review detailed information about the Flink Task Managers and Job Managers such as the memory model, memory consumption, configuration parameters and logs.

For Task Managers



For Job Managers



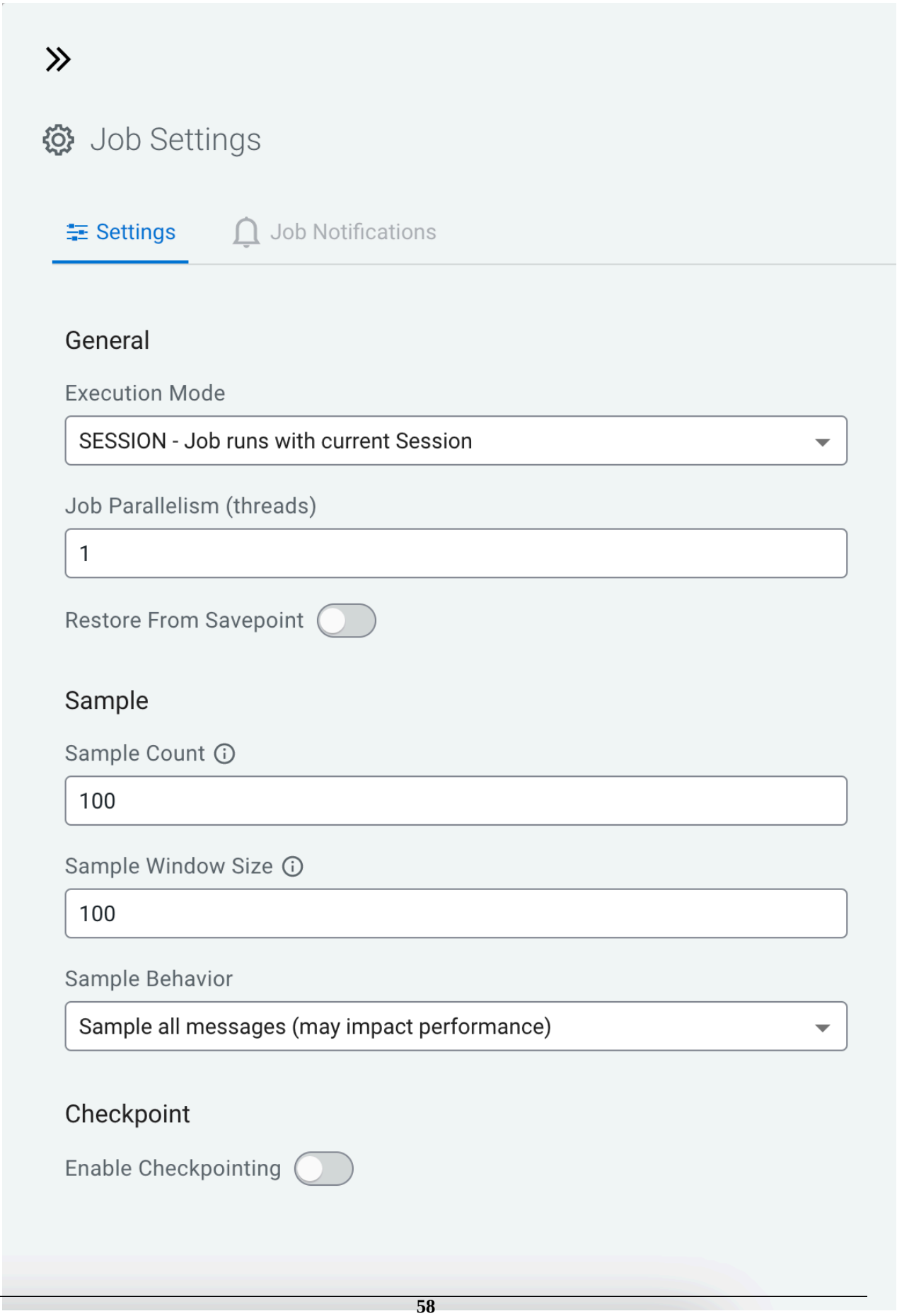
Managing SQL jobs

You can configure your SQL jobs with Flink features in the Streaming SQL Console or through REST API using the API Explorer.

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.


Job Settings
Settings
Job Notifications
General
Execution Mode
SESSION - Job runs with current Session
Job Parallelism (threads)
1
Restore From Savepoint
Sample
Sample Count
100
Sample Window Size
100
Sample Behavior
Sample all messages (may impact performance)
Checkpoint
Enable Checkpointing

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.

Sample Count

The number of sample entries shown under the Results tab. To have an unlimited number of sample entries, add 0 to the Sample Count value.

Sample Window Size

The number of sample entries to keep in under the Results tab. To have an unlimited number of sample entries, add 0 to 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

Restore From Savepoint

You can enable or disable restoring a SQL job from a Flink savepoint after stopping it.

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.

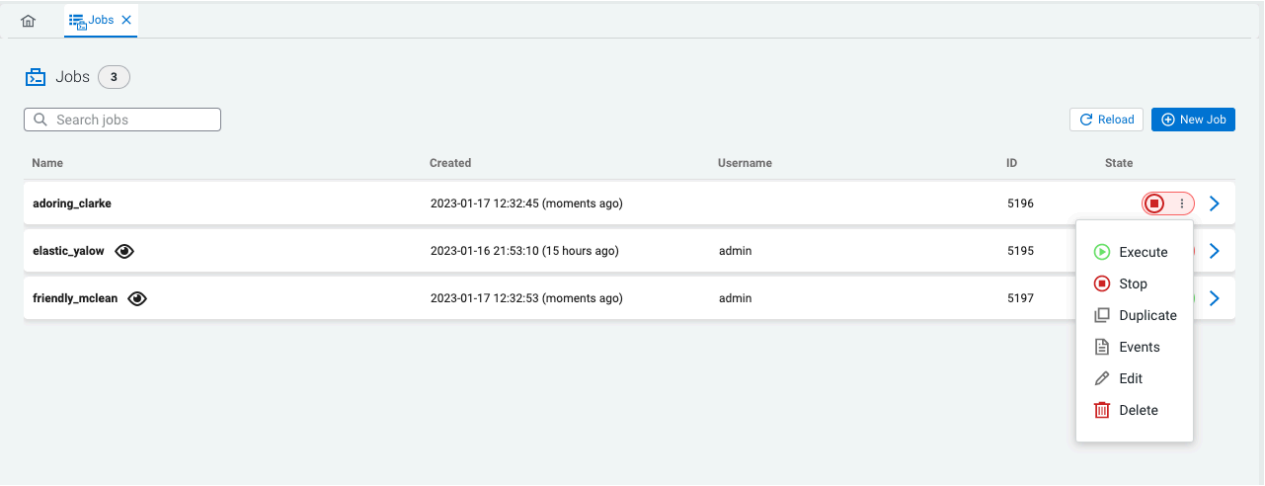
Failure Restart Strategy

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.

Managing SQL jobs

You can manage your SQL jobs and reload them regardless of success to the SQL Editor to make further and additional changes, and execute them again.

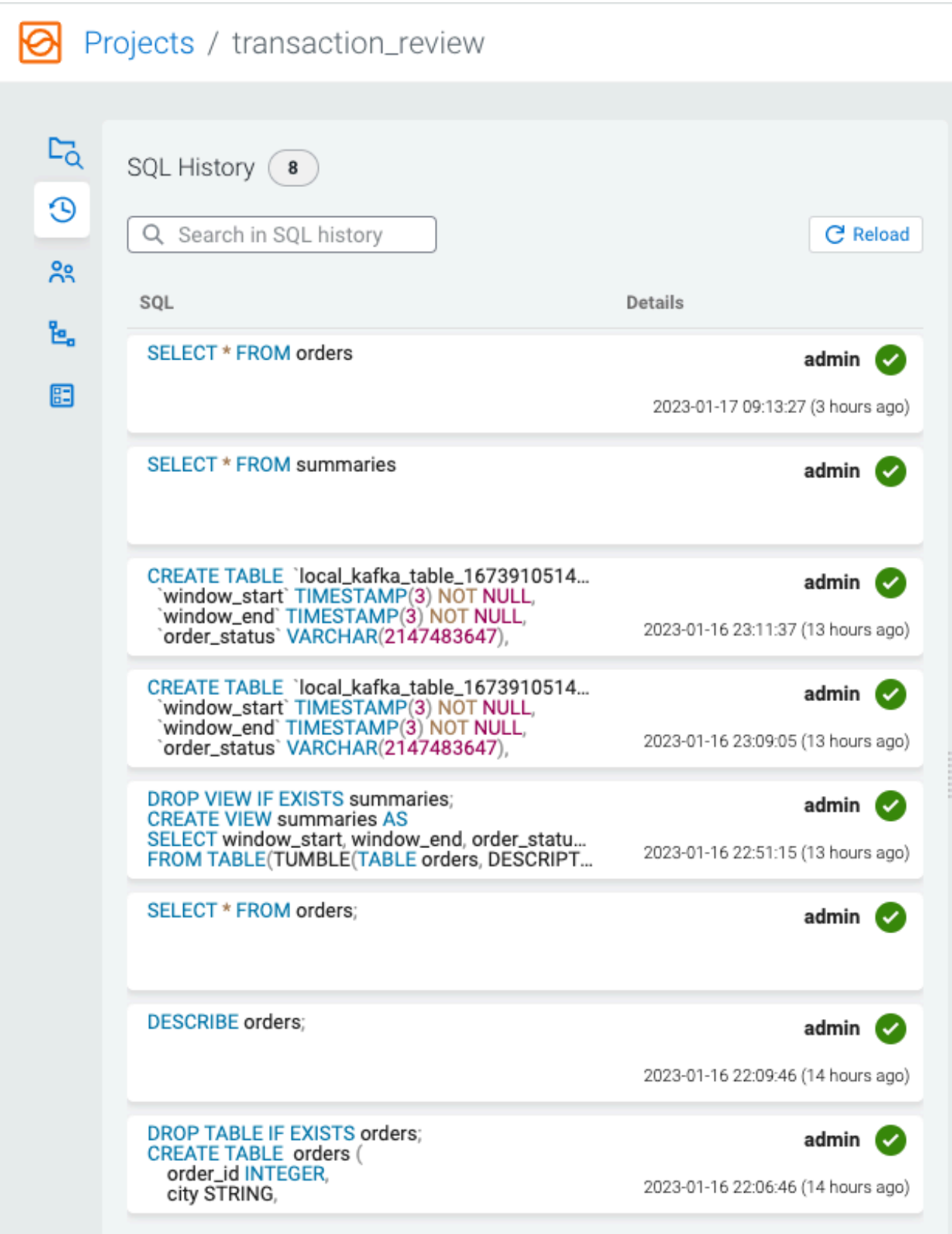
You can manage the SQL Jobs on the **Jobs** tab if you need to stop them, restart them, delete them, duplicate them, review their events and also reload them to the **SQL Editor** in case any change is needed to their configuration. To reload a SQL job to the SQL Editor, you can simply click on it from the list of jobs or use ⋮ Edit . The details of the job are reloaded to the SQL Editor with the latest saved configuration and Materialized Views.



Loading SQL queries to SQL Editor

You can reuse an already submitted SQL query from SQL History view of Streaming SQL Console.

When you need to reuse a SQL query for a new job, you can load it and execute it in the **SQL Editor**. You can use the SQL History view of Streaming SQL Console.



You can view the full submitted statement by hovering the mouse over the items. You can load the query to the editor by simply clicking on the query and the statement will appear in the **SQL Editor** next to it.

Using 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 SQL Stream Builder (SSB) REST API in command line, import them to REST API Tools or by accessing the Swagger UI.

The following HTTP methods are available for SSB:

- 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 SQL Stream Builder.

You can also reach the SSB REST API reference document from the main menu of Streaming SQL Console using the **API Explorer**.

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

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.