

Cloudera Streaming Analytics 1.5.0

CSA Community Edition

Date published: 2019-12-17

Date modified: 2021-09-29

CLOUdera

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

Legal Notice

© Cloudera Inc. 2025. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Community Edition Overview.....	4
Architecture.....	4
Release Notes.....	5
Installing CSA Community Edition.....	6
System Prerequisites.....	7
Community Edition Quickstart Guide.....	7
Creating a table.....	7
Running a SELECT query.....	10
Creating and querying a view.....	11
Adding a sink using a template.....	12
Adding a Materialized View.....	13
Troubleshooting.....	17

Community Edition Overview

The Community Edition of Cloudera Streaming Analytics (CSA) is a standalone deployment of Apache Flink and SQL Stream Builder (SSB). You can use the dockerized version of CSA to quickly setup Flink and SSB in your local environment, and try out the real-time streaming capabilities of Flink accompanied by the real-time querying features of SSB using SQL.

Architecture

The Community Edition of Cloudera Streaming Analytics consists of preconfigured Docker images for Zookeeper, Kafka and PostgreSQL to make getting started easier. The components can be reached using their dedicated ports. Storage for the Community Edition is handled by docker volumes, while PostgreSQL is integrated for database management and storing the Materialized Views.

PostgreSQL is used by SQL Stream Builder components internally. It is also used as the underlying database for the Materialized View Engine. The PostgreSQL database for the Materialized View tables (*EVENTADOR_SNAPPER* database) can be accessed by using the user *EVENTADOR_SNAPPER*. The default password for the database is *CLOUDERA*.

The containers use the following docker volumes to provide persistent local storage between restarts. If the volumes do not exist in your local environment, they are created when running the docker-compose up command.

ssb-volume

Persistent in the Flink TaskManager and JobManager containers. It is used for storing savepoints of the jobs. When using the Filesystem connector, it is also recommended to use a volume.

pg-volume

Used by the PostgreSQL database. It stores the internal tables required for SQL Stream Builder to work, as well as the created Materialized Views.

kf-volume

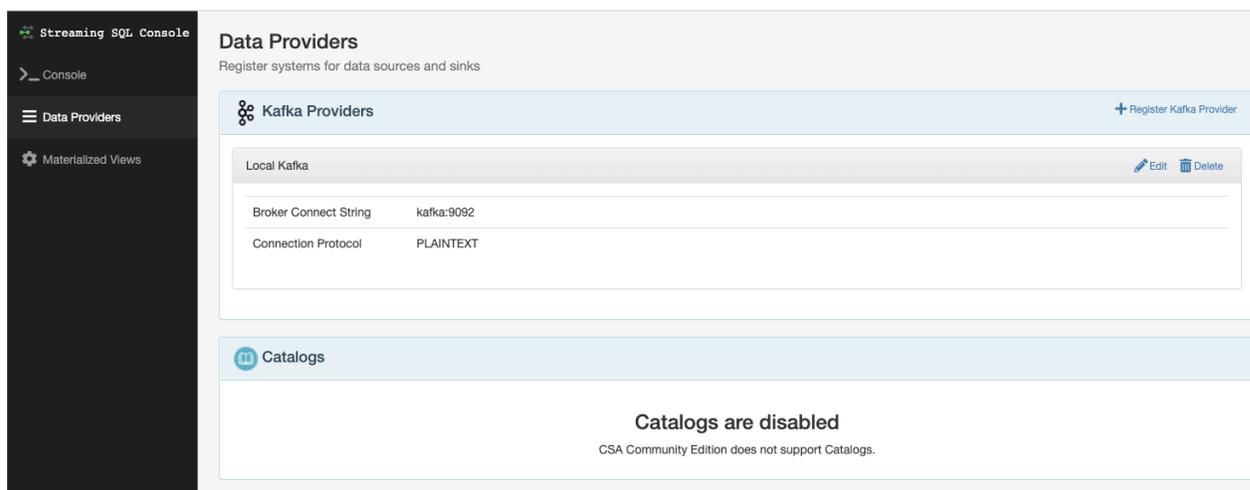
Used by the Kafka container to store the topics.

zk-volume

Used by Zookeeper.

It is possible to delete the docker volumes for a fresh start by shutting down all of the containers with `docker-compose down --volumes` command, or individually removing them with `docker volume rm <VOLUME NAME>` command.

By default, the Kafka container is preconfigured in SQL Stream Builder as the **Local Kafka** data provider.



For local prototyping to the preconfigured Kafka, you can create and produce data to topics with the following commands:

```
docker-compose exec kafka /opt/kafka/bin/kafka-topics.sh --bootstrap-server
kafka:9092
    --create --topic myNewTopic
```

```
docker-compose exec -T kafka /opt/kafka/bin/kafka-console-producer.sh --boot
strap-server
kafka:9092 --topic airplanes
```

Release Notes

Learn more about the supported component versions, known issues, fixed issues and limitations that apply to the Cloudera Streaming Analytics Community Edition.

Component version support

Apache Flink 1.13 is supported

Unsupported Features

- Kerberos authentication
- Catalog support

Limitations

The following limitations are applied to the Community Edition:

- Only 3 SQL jobs can be run simultaneously

- All jobs are submitted to the same standalone cluster.
- Storage option is only the docker volumes.
- Kafka and Filesystem to Docker volume are the only supported connectors.

Installing CSA Community Edition

You need to access the Downloads Page of Cloudera Streaming Analytics (CSA) to receive the Community Edition version of CSA. After receiving the docker file of CSA Community Edition, you need to create a composed version of the images and run CSA using Docker. The Streaming SQL Console is available using a local host address.

Before you begin

- For Mac/Windows users, make sure that you have Docker Desktop on your computer.
- Make sure that you have set the minimum memory and CPU for Docker. For more information, see the [System Prerequisites](#) section.

Procedure

1. Access the [Downloads Page of CSA Community Edition](#) to acquire the Community Edition artifact.
2. Create a folder to where you will download the Community Edition artifact.
3. Run the following command in command line after accessing the folder where you store the Community Edition artifact.

```
docker-compose up
```



Tip: When starting the containers, you can set the Flink TaskManager containers to 2 to have more resources when running SSB.

```
docker-compose up -d --scale flink-taskmanager=2
```

It can take time to start the Community Edition for the first time or when there are changes to the docker images.

4. Access the Streaming SQL Console after everything is up and running with the following local host address.

```
http://localhost:8000
```

5. Sign in to the Streaming SQL Console.

You have the option to create a new account or use the default username and password.

- Default username: admin
- Default password: admin

After providing the username and password, click Login.

As another option, you can also register a new account:

- a) Click Create account.
- b) Provide your Name, Username and Password.
- c) Click Register.
- d) Log in using the previously set Username and Password.

Results

You have installed the Community Edition of CSA on your local environment.

What to do next

SQL Stream Builder is ready to be used.

System Prerequisites

There are minimum system requirements that are need to be met when running the Community Edition of Cloudera Streaming Analytics (CSA).

Table 1: System prerequisites

Docker-compose version	1.6.0 or higher
Docker engine version	1.10.0 or higher
Memory	4 GB or more
CPU	4 cores or more
Required TCP ports	<ul style="list-style-type: none"> • Kafka: 9092 • PostgreSQL: 5432 • ZooKeeper: 2181 • Streaming SQL Console: 8000 • Streaming SQL Engine: 18121 • Materialized View Engine: 18131 • Flink Dashboard: 8081

Linux

The Community Edition should run on any Linux distribution that has a working docker and docker-compose installation. No configuration changes are typically required.

MacOS/Windows

MacOS and Windows users should use Docker Desktop to run SQL Stream Builder.

By default, Docker Desktop limits resources to containers. To run SQL Stream Builder, you need to select Docker -> Preferences , and increase the memory and CPU allocations.

Community Edition Quickstart Guide

You can use the Community Edition Quickstart Guide to learn and try the basic steps to use SQL Stream Builder through the Streaming SQL Console. The Quickstart Guide details the steps of how to create a table, run a query, create a view and add a Materialized View.

Creating a table

As a first step to use the Community Edition, learn how to create a table that generates random data using the included Faker connector after accessing the Streaming SQL Console.

When you access the Streaming SQL Console, you are redirected to the homepage of the SQL Stream Builder user interface. The homepage of Streaming SQL Console is the **Console page**. On the **Console page**, you can easily create and submit your SQL queries in the **SQL window**.

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 window**.

Use the following CREATE TABLE statement, and paste it to the **SQL window**.

```
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|DELIVERING|DELIVERED|CANCELED){1}'}'
);

```

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

The screenshot displays the Cloudera Streaming Analytics Console interface. On the left is a sidebar with navigation options: Console, Data Providers, and Materialized Views. The main area is titled "Console - Run SQL against streams of data and create persistent SQL streaming jobs". It features tabs for Compose, Tables, Functions, History, and SQL Jobs. The "SQL Job Name" field is set to "hopeful_brown". Below this, there are tabs for SQL, Materialized View, Settings, and Session. The SQL tab is active, showing a code editor with the following SQL statement:

```

1 DROP TABLE IF EXISTS orders;
2 CREATE TABLE orders (
3   order_id INTEGER,
4   city STRING,
5   street_address STRING,
6   amount INTEGER,
7   order_time TIMESTAMP(3),
8   order_status STRING,
9   WATERMARK FOR `order_time` AS order_time - INTERVAL '15' SECOND
10 ) WITH (
11   'connector' = 'faker',
12   'rows-per-second' = '1',
13   'fields.order_id.expression' = '#{number.numberBetween '0','99999999'}',
14   'fields.city.expression' = '#{Address.city}',
15   'fields.street_address.expression' = '#{Address.street_address}',
16   'fields.amount.expression' = '#{number.numberBetween '0','100'}',
17   'fields.order_time.expression' = '#{date.past '15','SECONDS'}',
18   'fields.order_status.expression' = '#{regexify '(RECEIVED|PREPARING|DELIVERING|DELIVERED|CANCELED){1}'}'
19 );
20

```

Below the code editor are controls for "Templates" (default mode, solarized dark), "Sample", "Stop", and "Execute" buttons. The "Execute" button is highlighted in green. Below the code editor, there are tabs for "Logs" and "Results". The "Results" tab is active, displaying the output of the SQL execution:

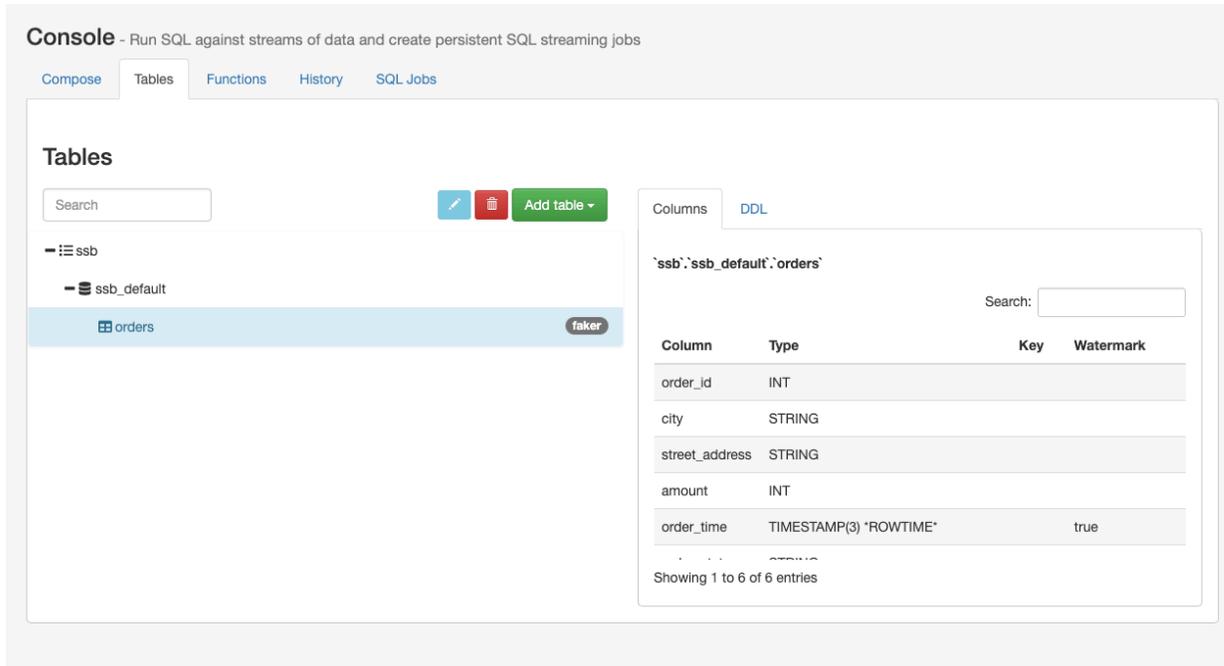
```

{date.past '15','SECONDS'}, fields.street_address.expression=#{Address.street_address}, schema.0.data-
type=INT, schema.2.name=street_address, fields.amount.expression=#{number.numberBetween '0','100'},
fields.order_id.expression=#{number.numberBetween '0','99999999'}, schema.1.name=city,
schema.4.name=order_time, schema.1.data-type=VARCHAR(2147483647), schema.3.data-type=INT, schema.2.data-
type=VARCHAR(2147483647), schema.3.name=amount, connector=faker, schema.watermark.0.rowtime=order_time,
schema.watermark.0.strategy.data-type=TIMESTAMP(3), schema.5.data-type=VARCHAR(2147483647),
fields.city.expression=#{Address.city}, rows-per-second=1, schema.4.data-type=TIMESTAMP(3),
fields.order_status.expression=#{regexify '(RECEIVED|PREPARING|DELIVERING|DELIVERED|CANCELED){1}'},
schema.0.name=order_id}, identifier: ['ssb`.`ssb_default`.`orders'], ignoreIfExists: [false], isTemporary:
[false]) command executed successfully.

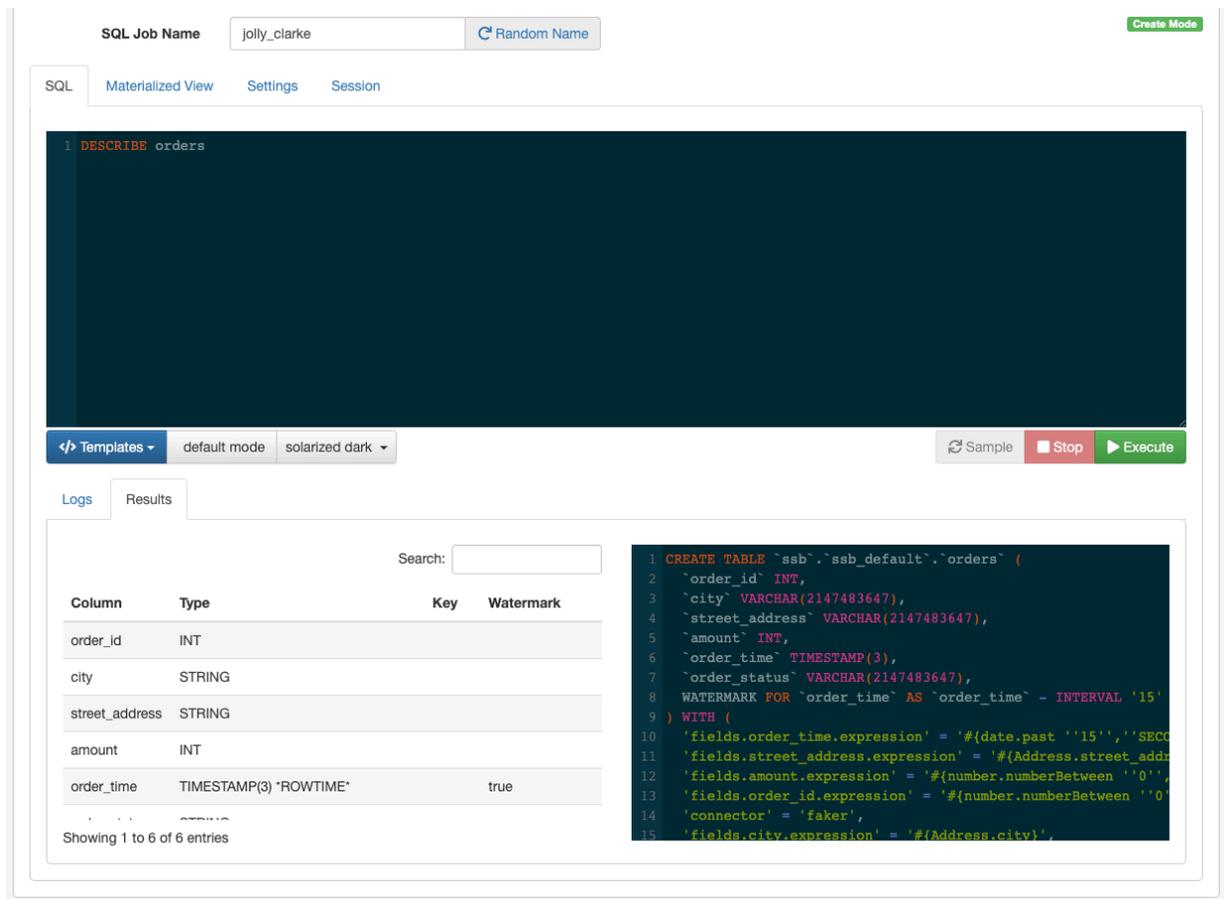
```

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

- You can view the table under the Tables tab:



- You can use the DESCRIBE orders; statement:



Running a SELECT query

After creating a table, learn how to run a SELECT query that outputs the resulted data under the Result tab of Streaming SQL Console.

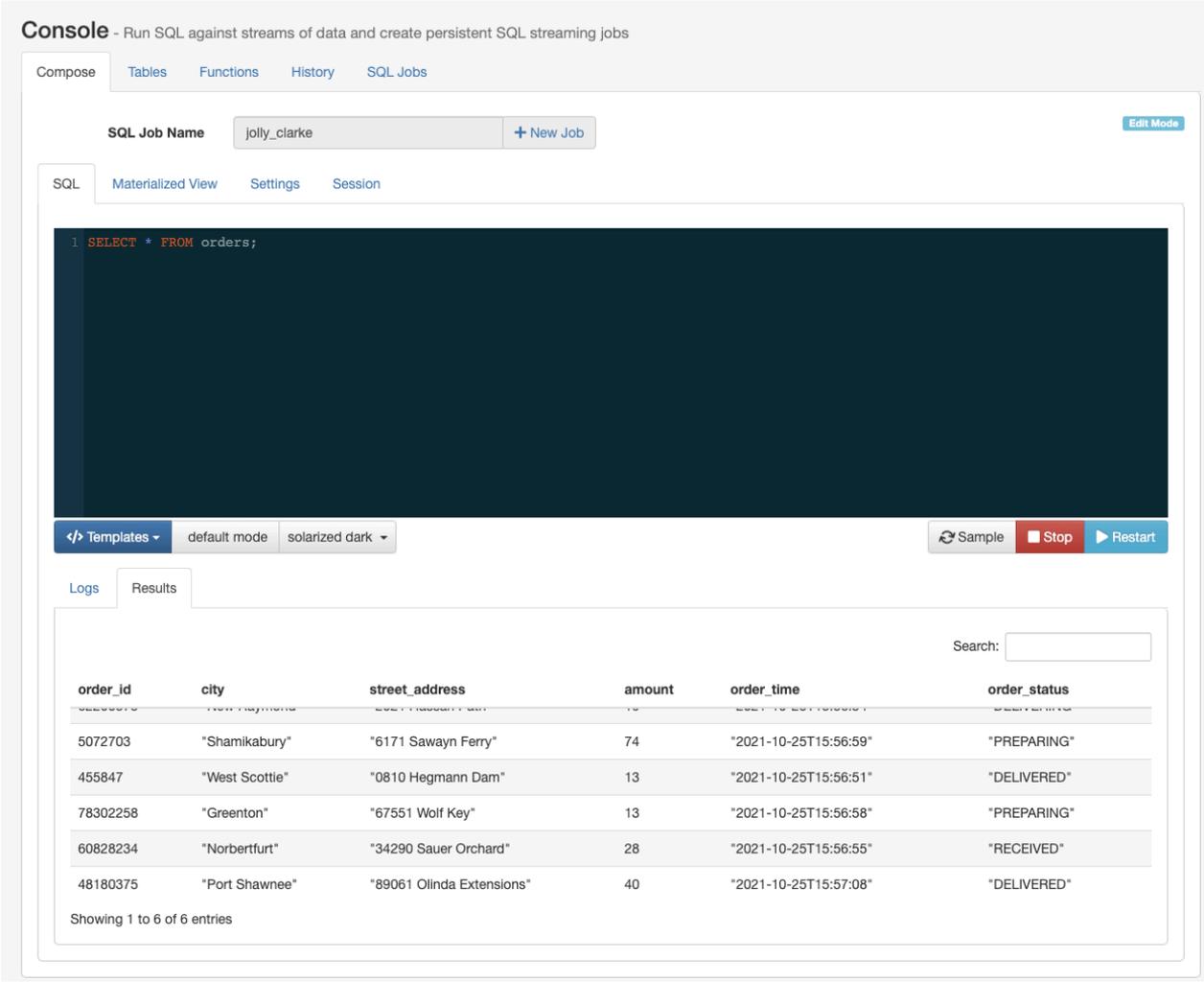
You have created your table using the Faker template that currently generates data. As a next step, you can run a SELECT query and output the results on the **Console page** under the **Results tab**.

Add the following SQL query to the **SQL window**:

```
SELECT * FROM orders;
```

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

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



The screenshot displays the Cloudera Streaming SQL Console interface. At the top, there's a header "Console - Run SQL against streams of data and create persistent SQL streaming jobs". Below this, there are tabs for "Compose", "Tables", "Functions", "History", and "SQL Jobs". The "SQL Jobs" tab is active, showing a "SQL Job Name" field with the value "jolly_clarke" and a "+ New Job" button. Below the job name, there are tabs for "SQL", "Materialized View", "Settings", and "Session". The "SQL" tab is active, showing a code editor with the query "1 SELECT * FROM orders;". Below the code editor, there are controls for "Templates" (default mode, solarized dark), "Sample", "Stop", and "Restart" buttons. At the bottom, there are tabs for "Logs" and "Results". The "Results" tab is active, showing a table of results with columns: order_id, city, street_address, amount, order_time, and order_status. The table contains 6 entries, with the first 5 visible. A search bar is located above the table. The text "Showing 1 to 6 of 6 entries" is displayed at the bottom of the results section.

order_id	city	street_address	amount	order_time	order_status
5072703	"Shamikabury"	"6171 Sawayn Ferry"	74	"2021-10-25T15:56:59"	"PREPARING"
455847	"West Scottie"	"0810 Hegmann Dam"	13	"2021-10-25T15:56:51"	"DELIVERED"
78302258	"Greenton"	"67551 Wolf Key"	13	"2021-10-25T15:56:58"	"PREPARING"
60828234	"Norbertfurt"	"34290 Sauer Orchard"	28	"2021-10-25T15:56:55"	"RECEIVED"
48180375	"Port Shawnee"	"89061 Olinda Extensions"	40	"2021-10-25T15:57:08"	"DELIVERED"

When you execute a query, a Flink job is created in the background. You can see more details about the created Flink job by navigating to the SQL Jobs tab and accessing the Flink Dashboard for the running job. You can also reload the SQL job into the **SQL window** to edit the job, or stop the job from the **SQL Jobs** tab.

Creating and querying a view

Now that you have learned the basic steps 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 create a view using the columns from the created Faker table. In this derived computation that can be saved as a view, the orders are aggregate in half minute intervals based on their status.

Use the following CREATE VIEW statement, and paste it to the **SQL window**.

```
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' SECONDS))
GROUP BY window_start, window_end, order_status;
```

To create the view, you need to click on the Execute button under the **SQL window**.

Before selecting data from this view, you can change the **Sample Behavior** to sample all messages on the **Settings** tabs. This way the sampled output is not limited to a given amount of messages.

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

```
SELECT * FROM summaries
```

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

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

The screenshot shows the Cloudera Streaming Analytics Console interface. At the top, it says "Console - Run SQL against streams of data and create persistent SQL streaming jobs". Below this, there are tabs for "Compose", "Tables", "Functions", "History", and "SQL Jobs". The "SQL Jobs" tab is active, showing a job named "jolly_clarke" with a "+ New Job" button and an "Edit Mode" button.

Under the job name, there are tabs for "SQL", "Materialized View", "Settings", and "Session". The "SQL" tab is active, showing a code editor with the query: "1 SELECT * FROM summaries;". Below the code editor, there are buttons for "Templates", "default mode", "solarized dark", "Sample", "Stop", and "Restart".

Below the code editor, there are tabs for "Logs" and "Results". The "Results" tab is active, showing a table with the following data:

Search:	window_start	window_end	order_status	order_count	total_amount
	"2021-10-25T15:57:40"	"2021-10-25T15:58"	"RECEIVED"	3	173
	"2021-10-25T15:58"	"2021-10-25T15:58:20"	"DELIVERED"	4	103

At the bottom of the results section, it says "Showing 1 to 2 of 2 entries".

Adding a sink using a template

Now that you know how to sample the queried results to the Result tab, learn how to create a sink using one of the predefined Templates to output the results of the executed SQL queries.

After some careful experimentation, you have executed the `SELECT * FROM summaries` query, stored it as a view and sampled the results to 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:

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

As the `OUTPUT_TABLE` 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 DDL` statement before the `SELECT` query. Before executing the queries, you must fill out the topic in the connector option.

Console - Run SQL against streams of data and create persistent SQL streaming jobs

Compose **Tables** Functions History SQL Jobs

SQL Job Name + New Job Create Mode

SQL **Materialized View** Settings Session

```

1 CREATE TABLE `ssb`.`ssb_default`.`output_table` (
2   `window_start` TIMESTAMP(3) NOT NULL,
3   `window_end` TIMESTAMP(3) NOT NULL,
4   `order_status` VARCHAR(2147483647),
5   `order_count` BIGINT NOT NULL,
6   `total_amount` INT
7 )
8 WITH (
9   `connector` = 'kafka: Local Kafka', -- Specify what connector to use, for the cluster-local (Docker) Kafka it must be 'kafka: Loc
10  `topic` = 'mytopic', -- To read data from when the table is used as source. It also supports topic list for source by separating to
11  `scan.startup.mode` = 'earliest-offset', -- Startup mode for Kafka consumer, valid values are 'earliest-offset', 'latest-offset', '
12  `format` = 'json' -- The format used to deserialize and serialize the value part of Kafka messages. Note: Either this option or the
13  -- `json.map-null-key.mode` = 'FAIL' -- Optional flag to control the handling mode when serializing null key for map data, FAIL by
14  -- `json.encode.decimal-as-plain-number` = 'false' -- Optional flag to specify whether to encode all decimals as plain numbers inst
15  -- `json.map-null-key.literal` = 'null' -- Optional flag to specify string literal for null keys when 'map-null-key.mode' is LITERA
16  -- `json.fail-on-missing-field` = 'false' -- Optional flag to specify whether to fail if a field is missing or not, false by default
17  -- `json.ignore-parse-errors` = 'false' -- Optional flag to skip fields and rows with parse errors instead of failing; fields are s
18  -- `json.timestamp-format.standard` = 'SQL' -- Optional flag to specify timestamp format, SQL by default. Option ISO-8601 will pars
19 );
20 INSERT INTO output_table SELECT * FROM summaries;

```

Templates

- blackhole
- datagen
- faker
- filesystem
- kafka
 - avro
 - csv
 - json
 - raw
- local-kafka
- print
- upsert-kafka

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

Adding a Materialized View

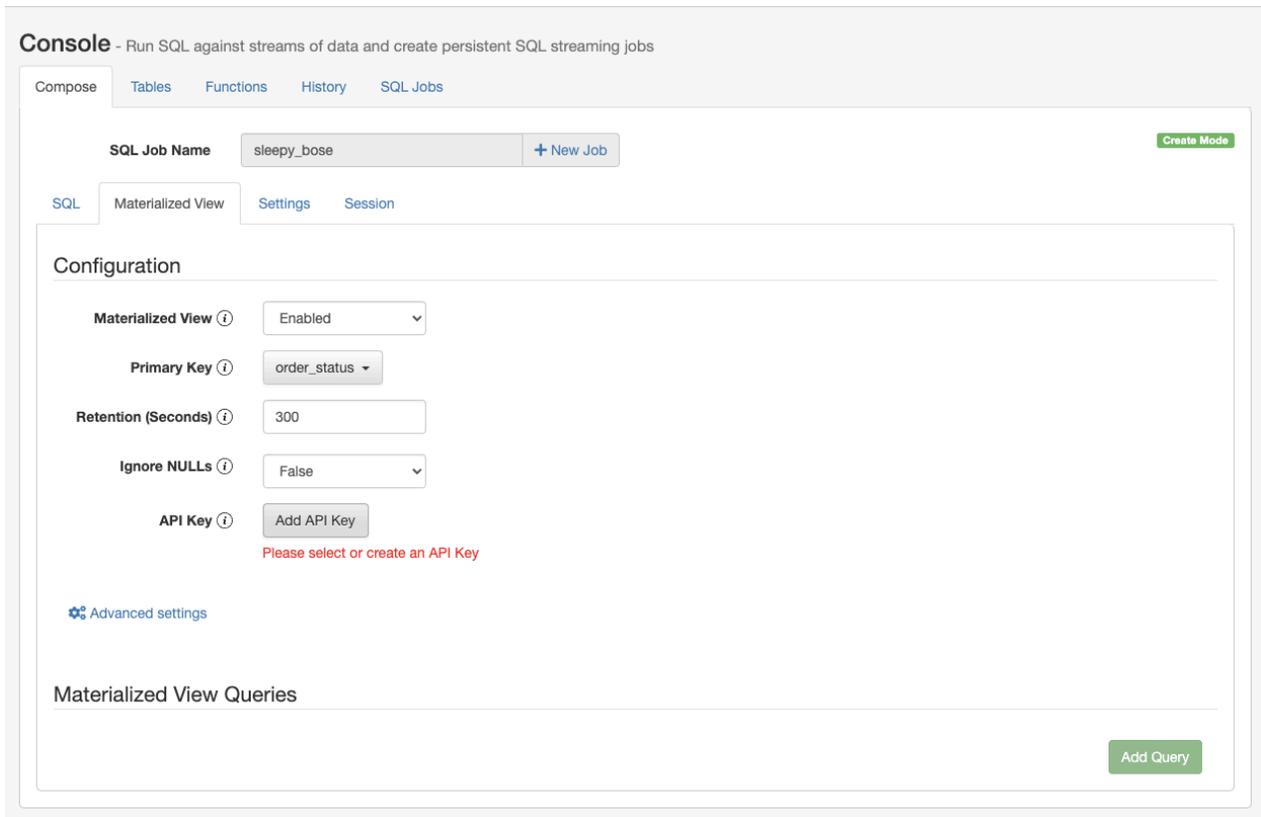
Materialized views let you create a persistent snapshot of the stream that can be queried through a REST endpoint. This can then be queried by external applications and tools. Now, that you know how to query simple and aggregated data, learn how to create Materialized Views.

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.

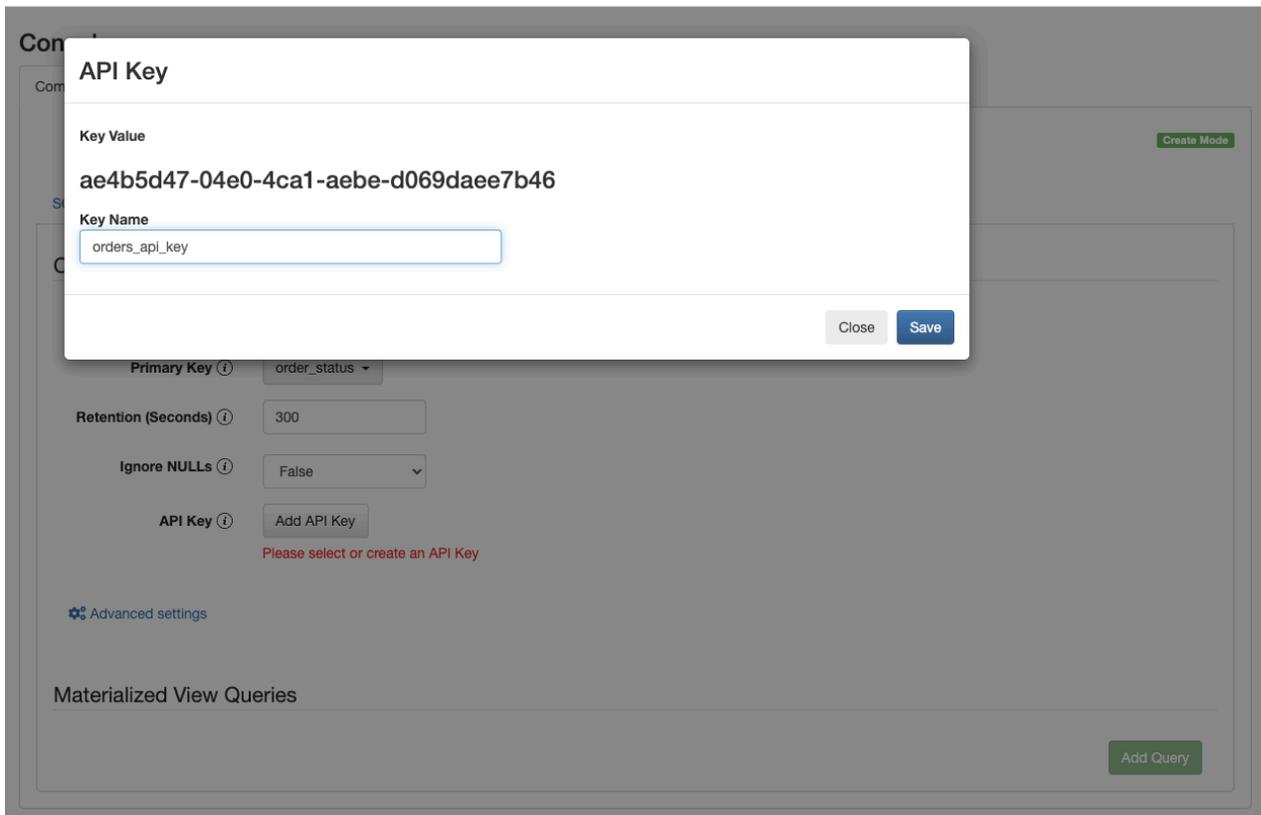
First, you need a New Job with a SELECT query. Add the following SQL query to the SQL window to run the query:

```
SELECT * FROM summaries
```

Before executing the query, you need to enable the Materialized View feature. On the Materialized View tab, 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.



After configuring the Materialized View, you can add a query. On the Query Configuration page, specify the URL 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 name of the URL, `order_status`, `order_count`, `total_amount` and `window_start` columns are selected as fields. A filter is also added to exclude the orders with a *CANCELED* status.

Materialized View Query Configuration

URL Pattern

`/api/v1/query/<Job ID>/` summary_query

Description (optional)

Query Builder

window_start

Add Column

Select All

Unselect All

Name	Alias	Type	Actions
order_status	order_status	VARCHAR	
order_count	order_count	BIGINT	
total_amount	total_amount	INTEGER	
window_start	window_start	TIMESTAMP_WITHOUT_TIME_ZONE	

Filters

AND OR

+ Add rule + Add group

order_status

not equal

CANCELED

✕ Delete

Close

Save Changes

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

When you click on the URL pattern, you are redirected to the 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 tab on the Streaming SQL Console, 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.



```

[
  - {
    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"
  }
]

```

Troubleshooting

When using the Community Edition and encounter an error, you can use the following examples to help you troubleshoot the issue.

'Port is already in use' error message

You need to verify that the required ports are available and not already in use.

Use one of the following commands to see which ports are in used:

- MacOS/Windows: netstat -an
- Linux: netstat -nlp or ss -nlp

exited 137 error message

If you see tmp-sqlio-1 exited with code 137 error message in the Streaming SQL Engine log, you need to the modify docker memory Resources settings.

Open Docker Desktop, go to the Settings > Resources , and set memory limit to 4 GB

If you are running a previous cluster and need to restart the Community Edition from scratch, you can run the following commands:

```

docker-compose down
docker rm -f $(docker ps -a -q)
docker volume rm $(docker volume ls -q)
docker-compose up -d

```