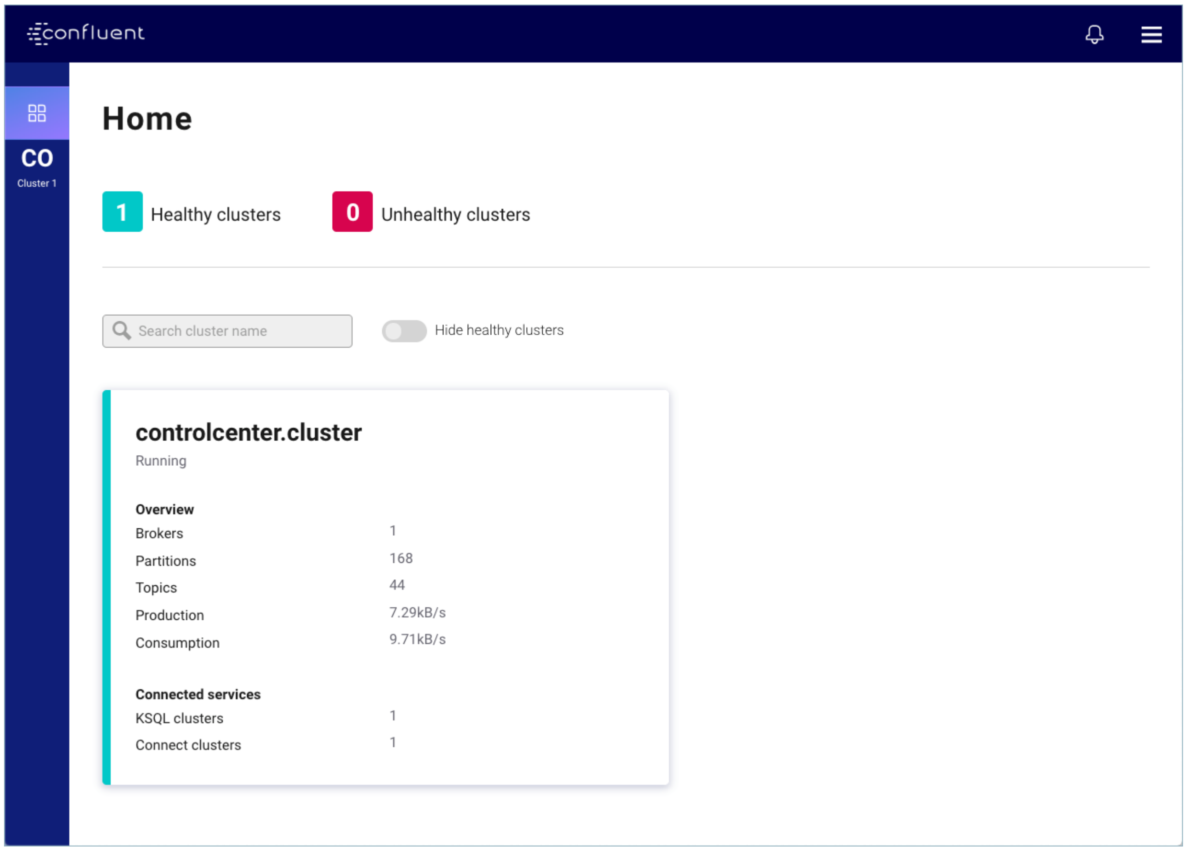
Create Kafka Topics

In this step, you create Kafka topics by using the Confluent Control Center. Confluent Control Center provides the functionality for building and monitoring production data pipelines and event streaming applications.

1. Navigate to the Control Center web interface at http://localhost:9021/.

Important

It may take a minute or two for Control Center to come online.



controlcenter.cluster
Running

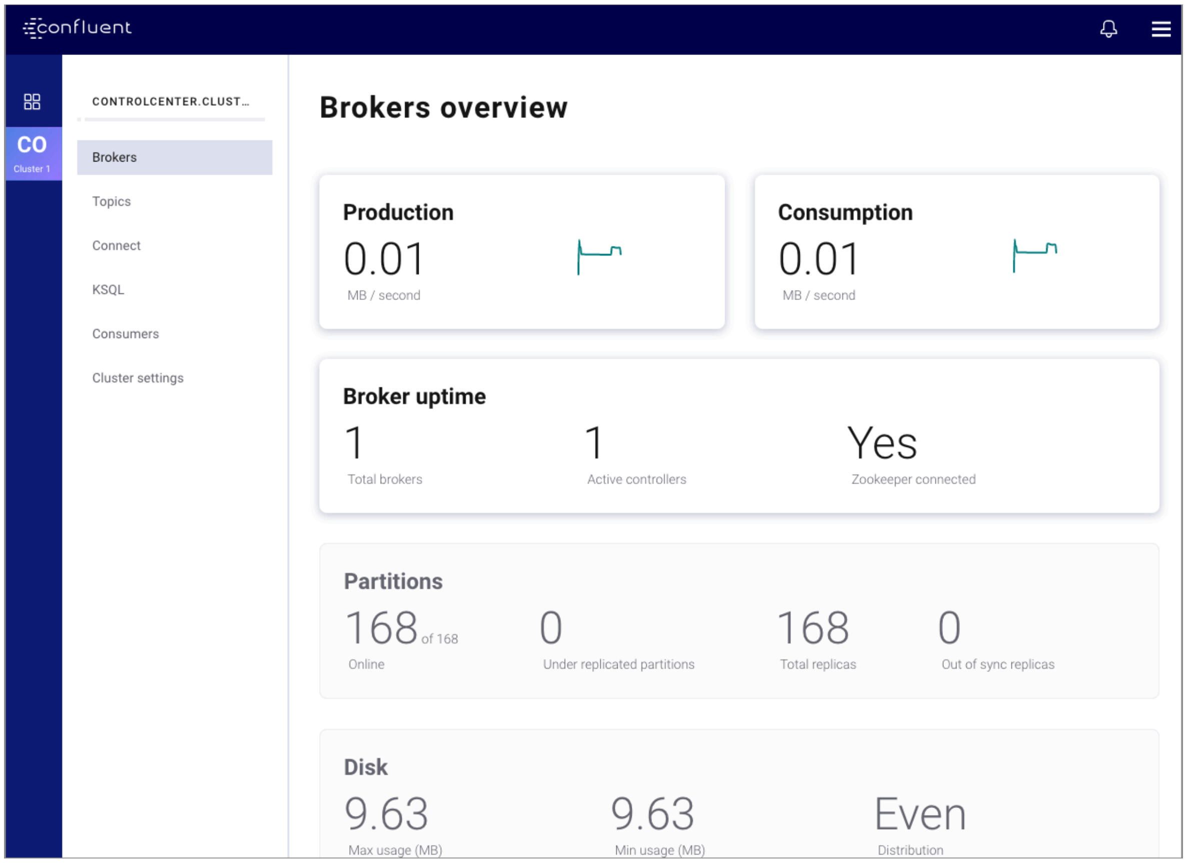
Overview

Brokers	1
Partitions	168
Topics	44
Production	7.29kB/s
Consumption	9.71kB/s

Connected services

KSQL clusters	1
Connect clusters	1

2. Select your cluster name.



Brokers overview

Production
0.01
MB / second

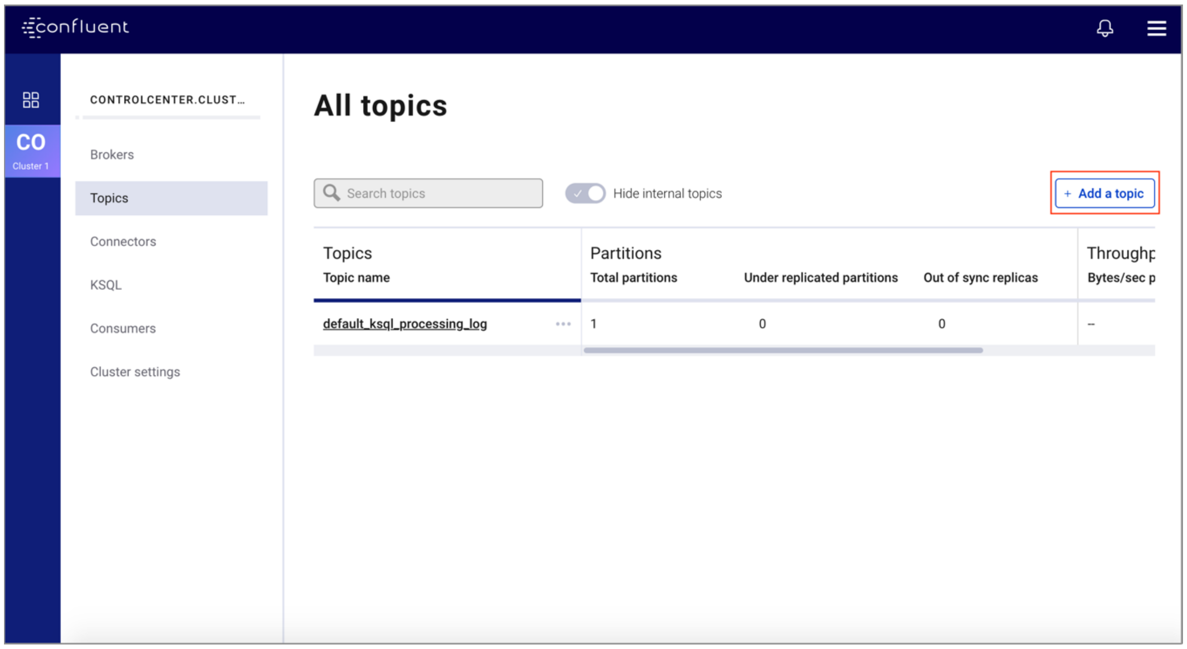
Consumption
0.01
MB / second

Broker uptime
1 Total brokers
1 Active controllers
Yes Zookeeper connected

Partitions
168 of 168 Online
0 Under replicated partitions
168 Total replicas
0 Out of sync replicas

Disk
9.63 Max usage (MB)
9.63 Min usage (MB)
Even Distribution

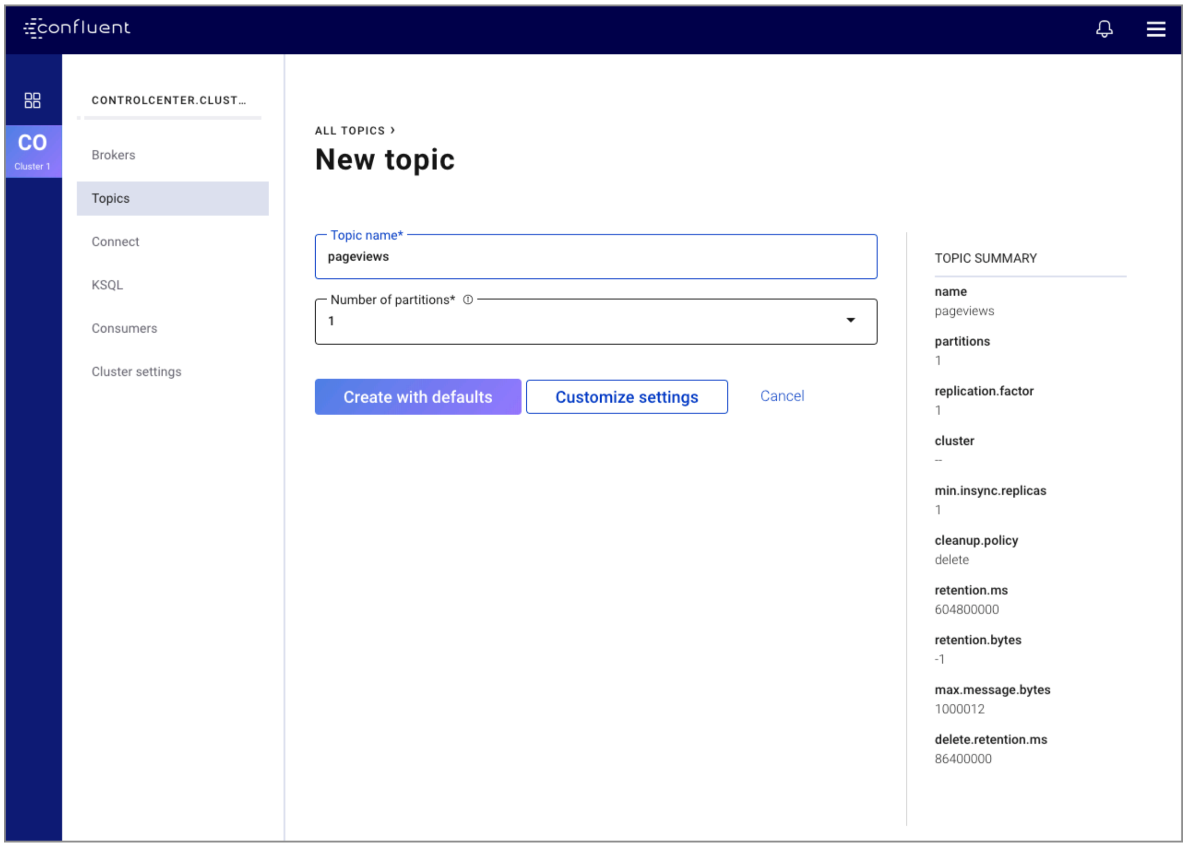
3. Select **Topics** from the cluster submenu and click **Add a topic**.



All topics

Topics	Partitions			Throughp
Topic name	Total partitions	Under replicated partitions	Out of sync replicas	Bytes/sec p
default_ksql_processing_log	1	0	0	–

4. Create a topic named `pageviews` and click **Create with defaults**.



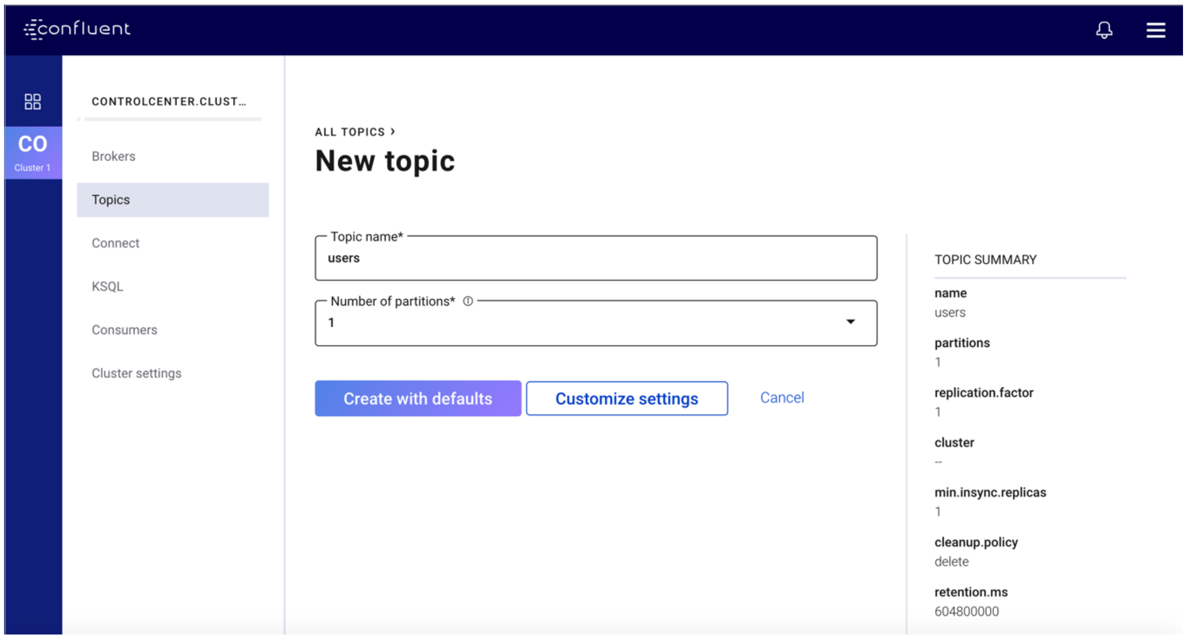
Topic name*: pageviews

Number of partitions*: 1

TOPIC SUMMARY

name	pageviews
partitions	1
replication.factor	1
cluster	--
min.insync.replicas	1
cleanup.policy	delete
retention.ms	604800000
retention.bytes	-1
max.message.bytes	1000012
delete.retention.ms	86400000

5. Repeat the previous steps and create a topic named `users` and click **Create with defaults**.



Topic name*: users

Number of partitions*: 1

TOPIC SUMMARY

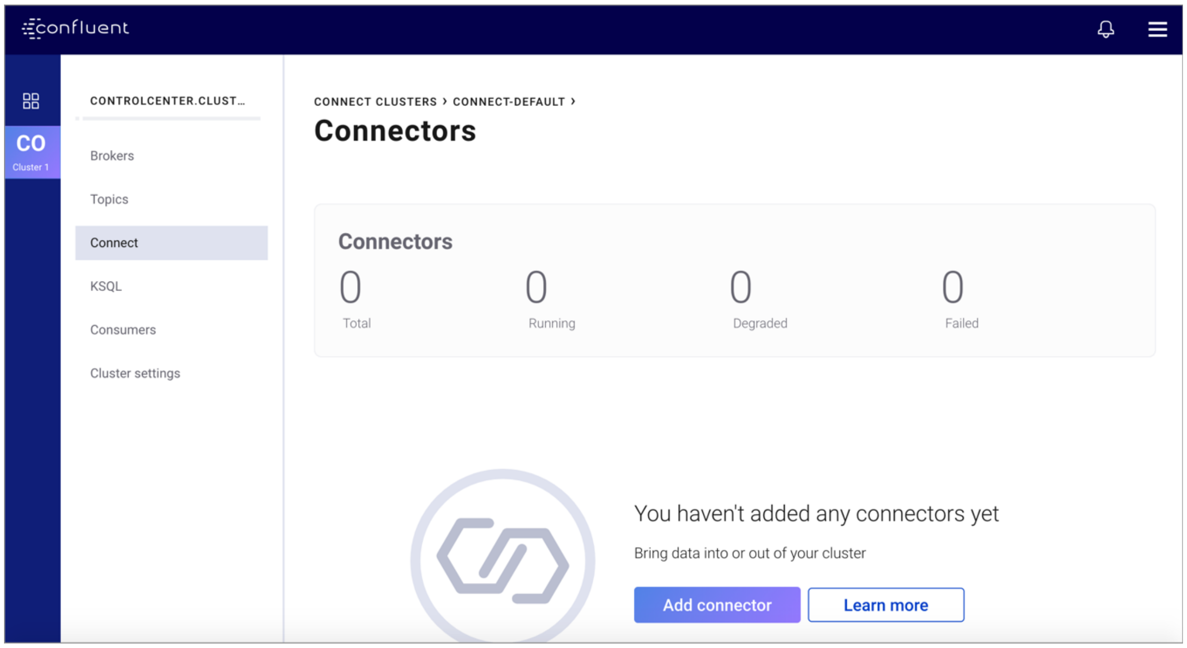
name	users
partitions	1
replication.factor	1
cluster	--
min.insync.replicas	1
cleanup.policy	delete
retention.ms	604800000

Install a Kafka Connector and Generate Sample Data

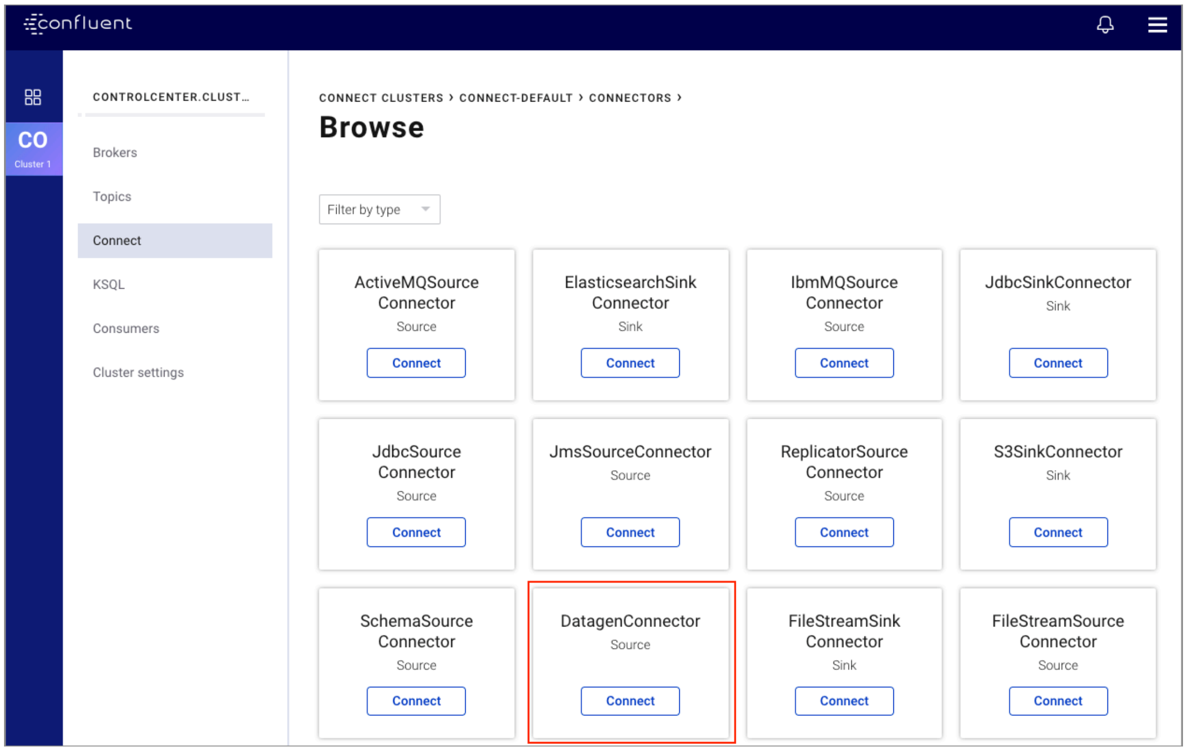
In this step, you use Kafka Connect to run a demo source connector called `kafka-connect-datagen` that creates sample data for the Kafka topics `pageviews` and `users`.

1. Run one instance of the Kafka Connect Datagen connector to produce Kafka data to the `pageviews` topic in AVRO format.

    1. From your cluster, click **Connect**.
    2. Select the `connect_default` cluster and click **Add connector**.

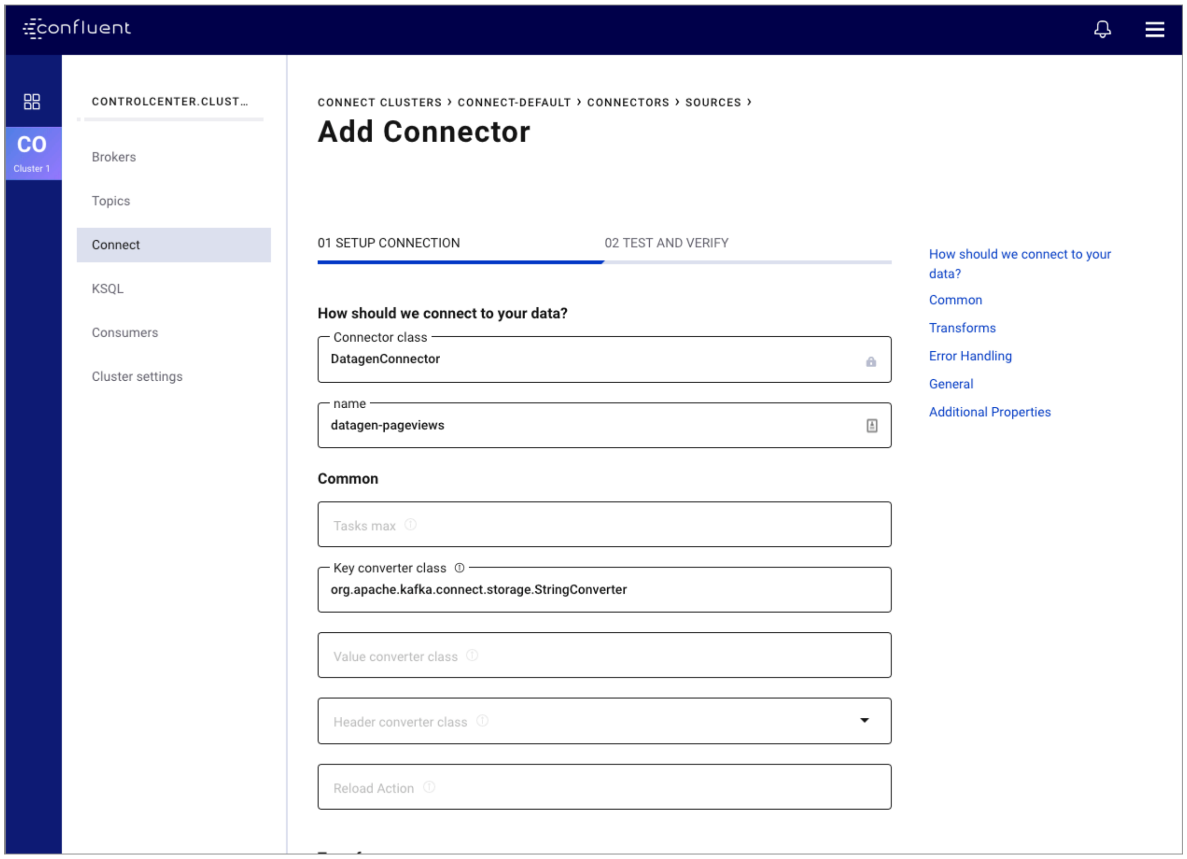


    3. Find the DatagenConnector tile and click **Connect**.


confluent
CO
Cluster 1
CONTROLCENTER.CLUST...
Brokers
Topics
Connect
KSQL
Consumers
Cluster settings
CONNECT CLUSTERS › CONNECT-DEFAULT › CONNECTORS ›
Browse
Filter by type
ActiveMQSource Connector Source Connect
ElasticsearchSink Connector Sink Connect
IbmMQSource Connector Source Connect
JdbcSinkConnector Sink Connect
JdbcSource Connector Source Connect
JmsSourceConnector Source Connect
ReplicatorSource Connector Source Connect
S3SinkConnector Sink Connect
SchemaSource Connector Source Connect
DatagenConnector Source Connect
FileStreamSink Connector Sink Connect
FileStreamSource Connector Source Connect

    4. Name the connector `datagen-pageviews`. After naming the connector, new fields appear. Scroll down and specify the following configuration values:

- In the **Key converter class** field, type `org.apache.kafka.connect.storage.StringConverter`.
- In the **kafka.topic** field, type `pageviews`.
- In the **max.interval** field, type `100`.
- In the **iterations** field, type `1000000000`.
- In the **quickstart** field, type `pageviews`.



Add Connector

How should we connect to your data?

Connector class: DatagenConnector

name: datagen-pageviews

Common

Tasks max

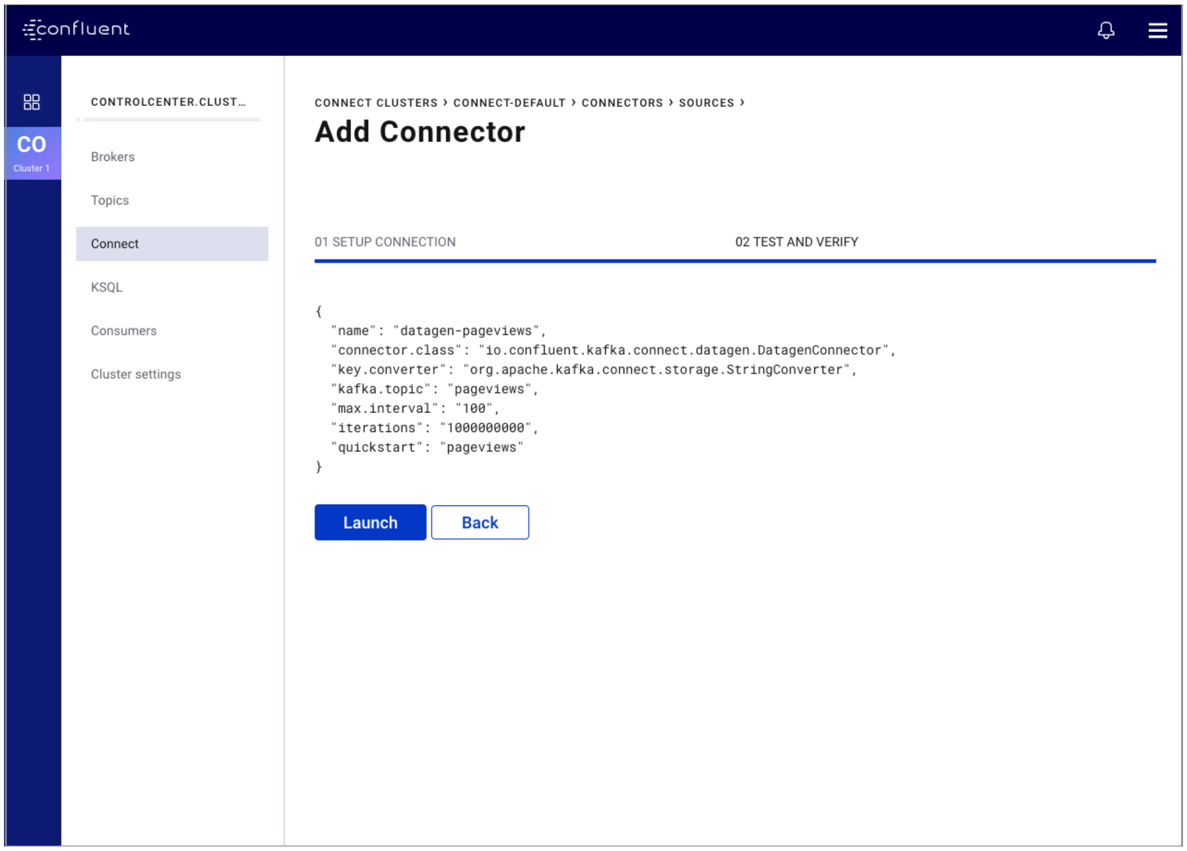
Key converter class: org.apache.kafka.connect.storage.StringConverter

Value converter class

Header converter class

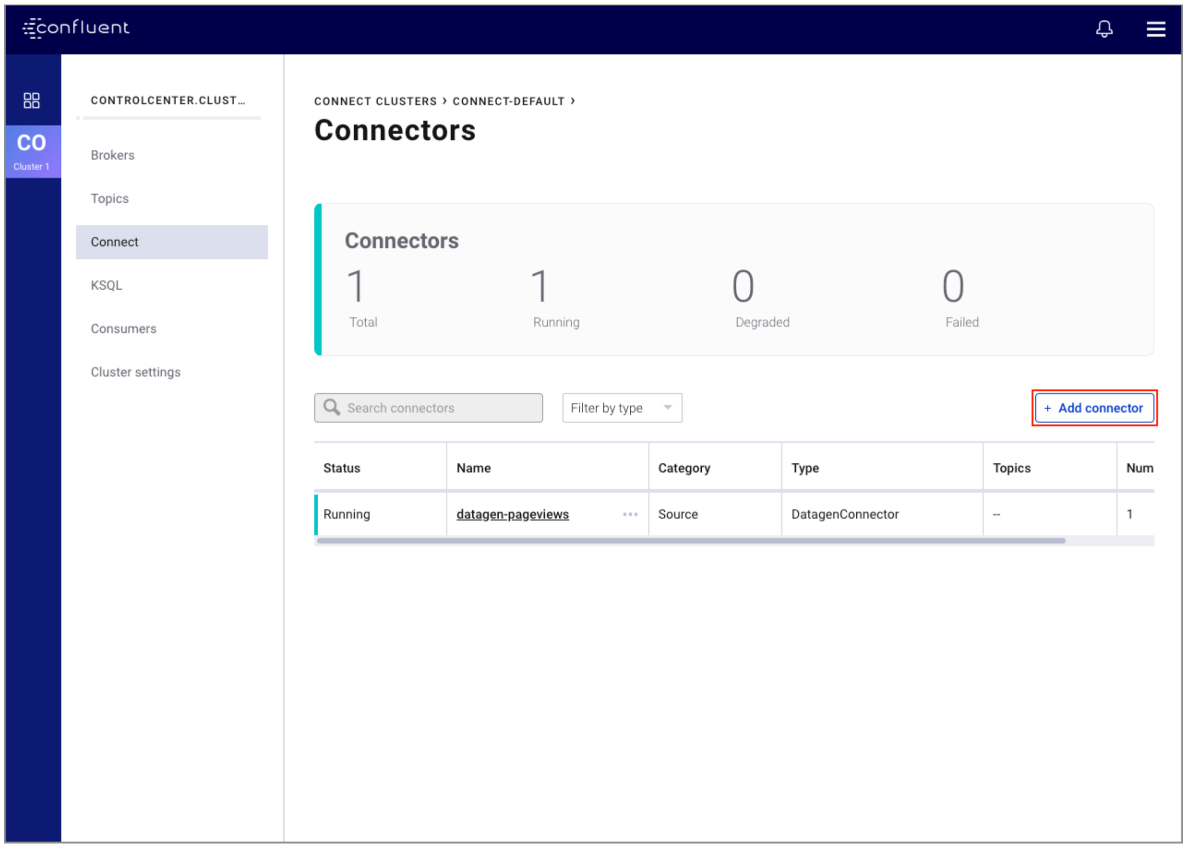
Reload Action

5. Click **Continue**.
6. Review the connector configuration and click **Launch**.



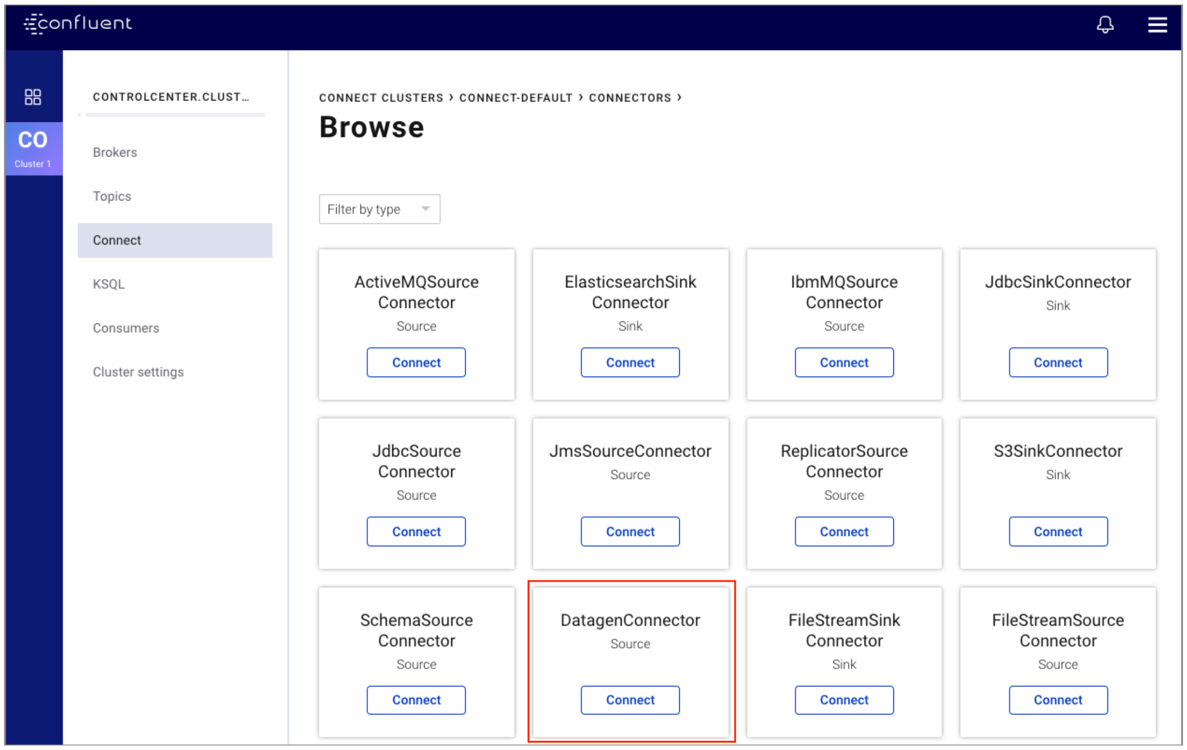
```
{
  "name": "datagen-pageviews",
  "connector.class": "io.confluent.kafka.connect.datagen.DatagenConnector",
  "key.converter": "org.apache.kafka.connect.storage.StringConverter",
  "kafka.topic": "pageviews",
  "max.interval": "100",
  "iterations": "1000000000",
  "quickstart": "pageviews"
}
```

2. Run another instance of the Kafka Connect Datagen connector to produce Kafka data to the `users` topic in AVRO format.
    1. Click **Add connector**.



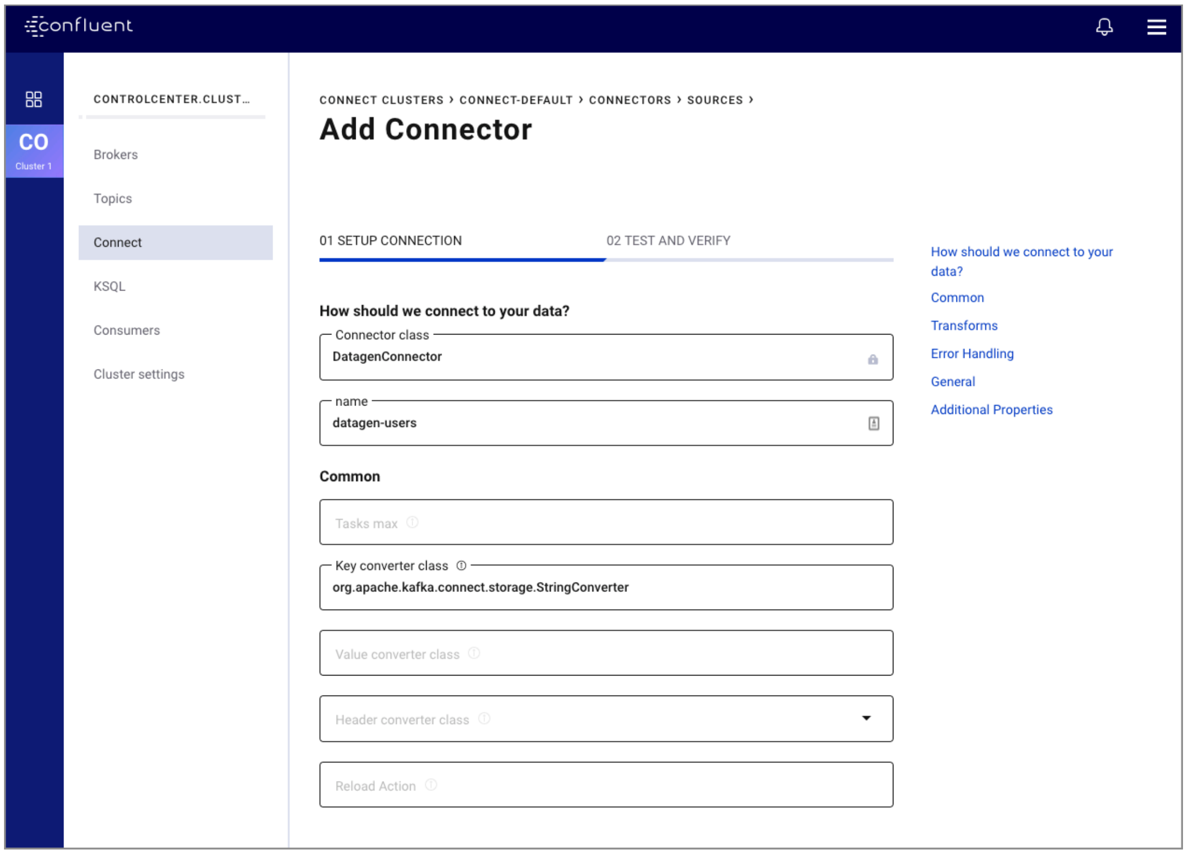
Status	Name	Category	Type	Topics	Num
Running	datagen-pageviews	Source	DatagenConnector	--	1

2. Find the DatagenConnector tile and click **Connect**.



3. Name the connector `datagen-users`. After naming the connector, new fields appear. Scroll down and specify the following configuration values:

- In the **Key converter class** field, type `org.apache.kafka.connect.storage.StringConverter`.
- In the **kafka.topic** field, type `users`.
- In the **max.interval** field, type `1000`.
- In the **iterations** field, type `1000000000`.
- In the **quickstart** field, type `users`.



Add Connector

01 SETUP CONNECTION　02 TEST AND VERIFY

How should we connect to your data?

Connector class: DatagenConnector

name: datagen-users

Common

Tasks max

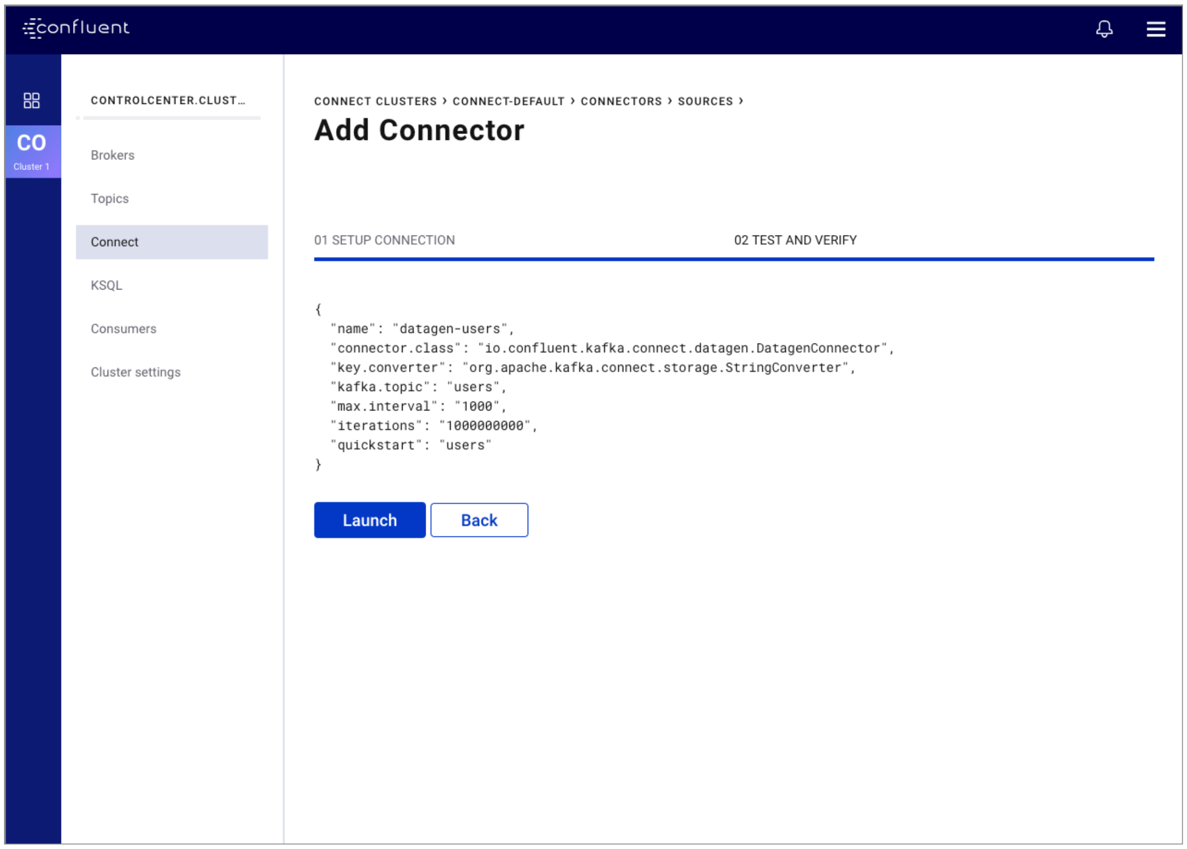
Key converter class: org.apache.kafka.connect.storage.StringConverter

Value converter class

Header converter class

Reload Action

4. Click **Continue**.
5. Review the connector configuration and click **Launch**.



Add Connector

01 SETUP CONNECTION　02 TEST AND VERIFY

```
{
  "name": "datagen-users",
  "connector.class": "io.confluent.kafka.connect.datagen.DatagenConnector",
  "key.converter": "org.apache.kafka.connect.storage.StringConverter",
  "kafka.topic": "users",
  "max.interval": "1000",
  "iterations": "1000000000",
  "quickstart": "users"
}
```

Create and Write to a Stream and Table using KSQL

In this step, KSQL queries are run on the `pageviews` and `users` topics that were created in the previous step. The KSQL commands are run using the KSQL tab in Control Center.

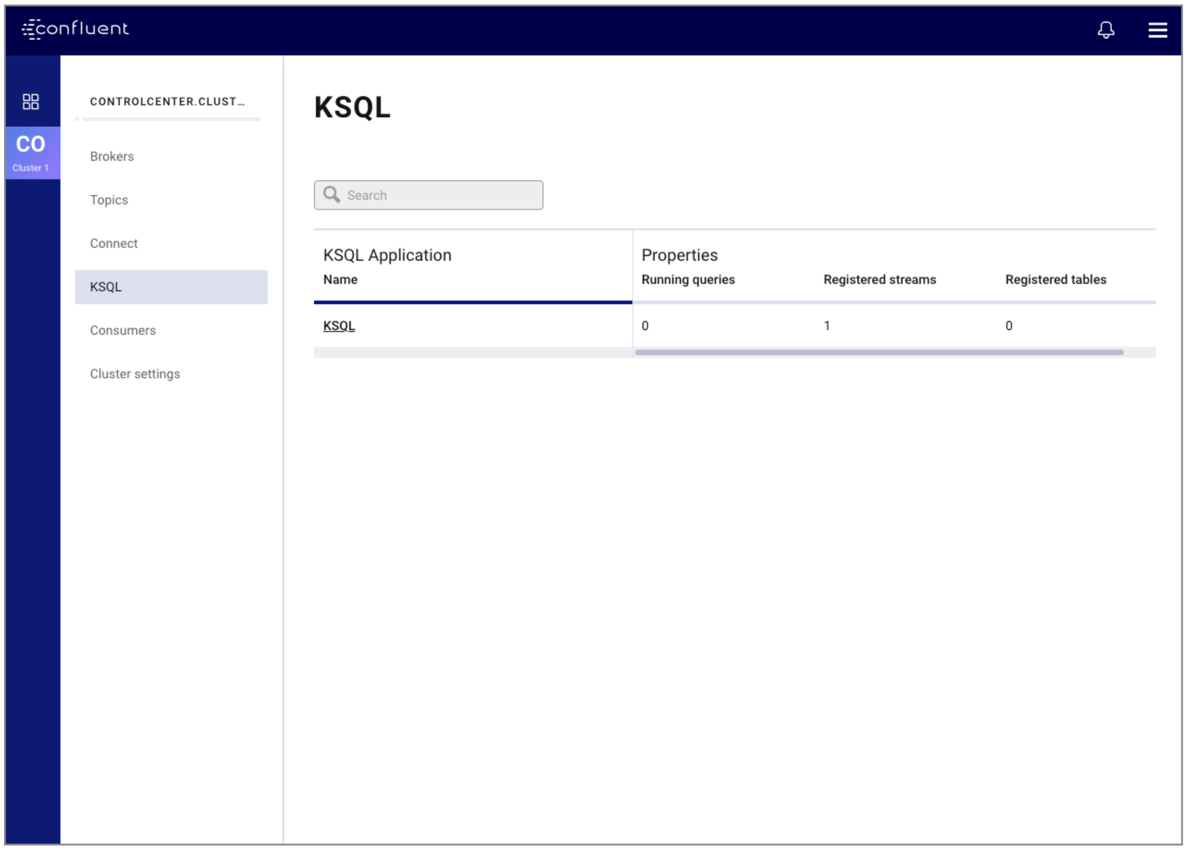
Tip

You can also run these commands using the KSQL CLI from your Docker container with this command: `docker-compose exec ksql-cli ksql http://ksql-server:8088`.

Create Streams and Tables

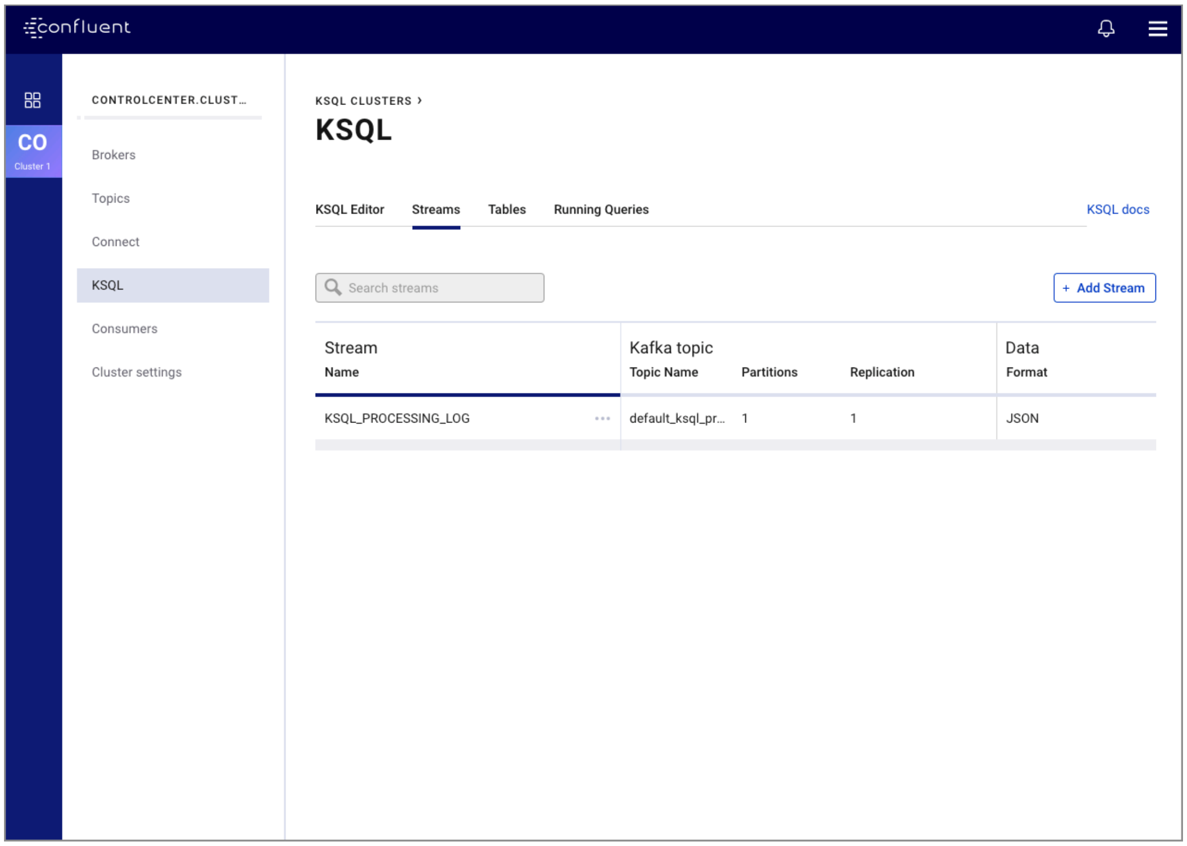
In this step, KSQL is used to create a stream for the `pageviews` topic, and a table for the `users` topic.

1. From your cluster, click **KSQL** and choose the **KSQL** application.



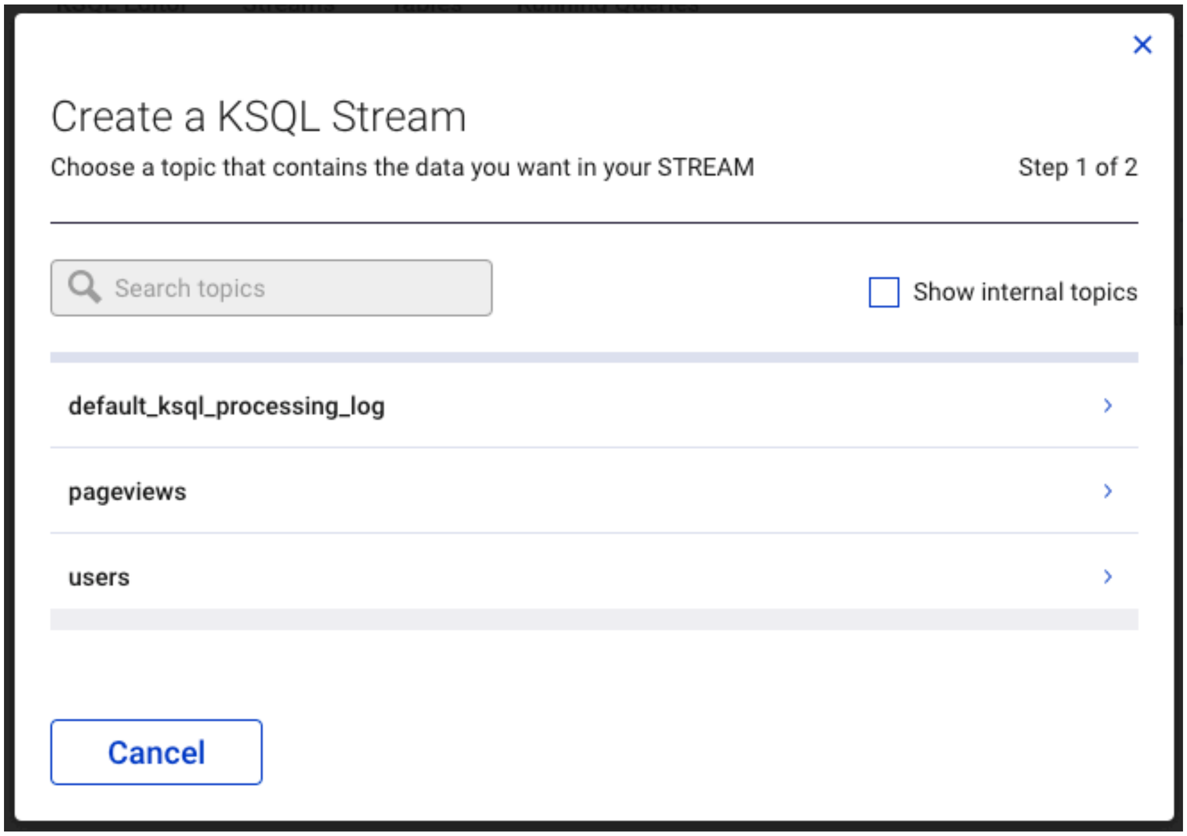
KSQL Application	Properties		
Name	Running queries	Registered streams	Registered tables
KSQL	0	1	0

2. From the **KSQL EDITOR** page, click the **Streams** tab and **Add Stream**.



Stream	Kafka topic			Data
Name	Topic Name	Partitions	Replication	Format
KSQL_PROCESSING_LOG	default_ksql_pr…	1	1	JSON

3. Select the `pageviews` topic.



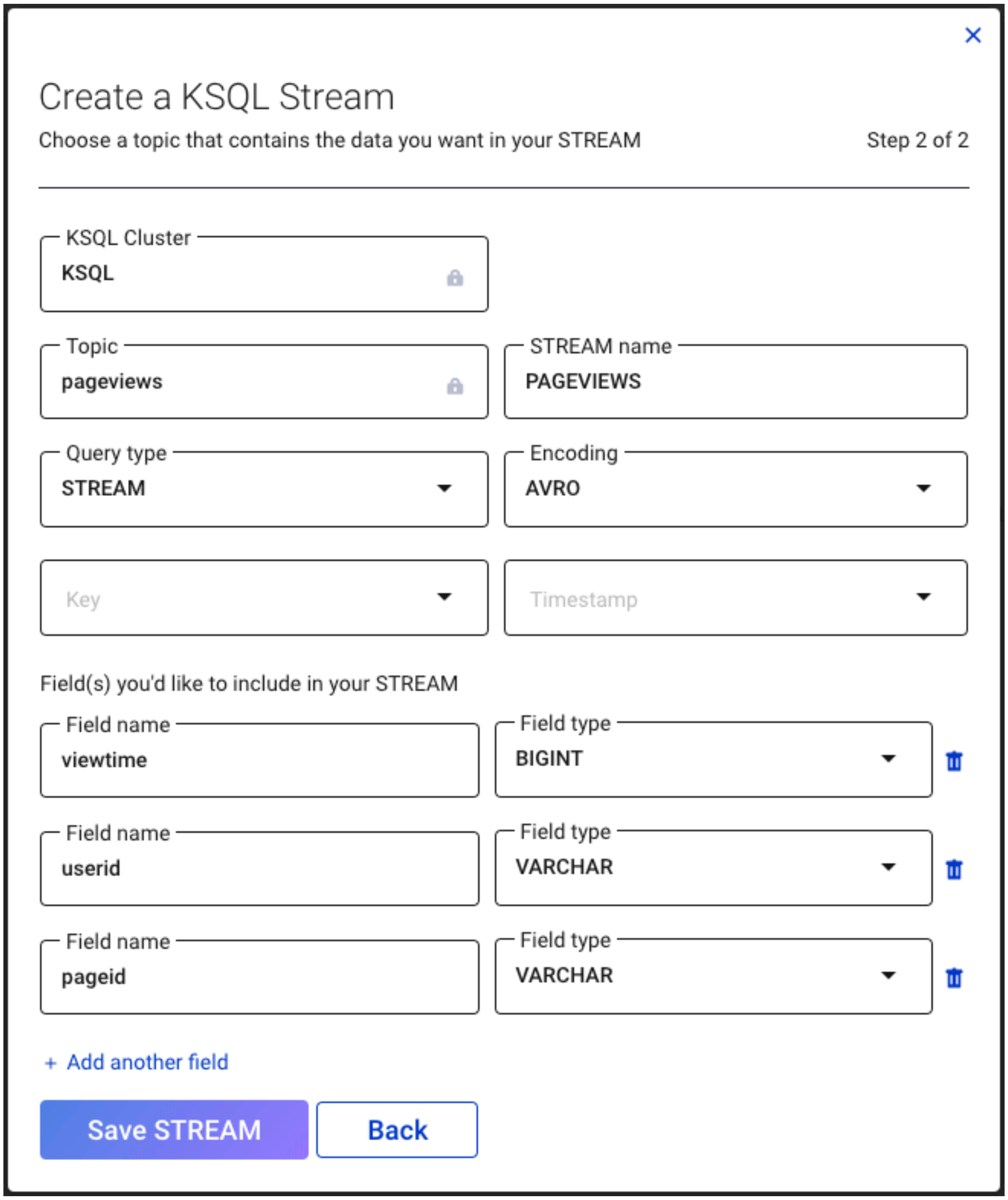
Choose a topic that contains the data you want in your STREAM

- default_ksql_processing_log
- pageviews
- users

4. Choose your stream options:

- In the **Encoding** field, select `AVRO`.

- In the **Field(s) you'd like to include in your STREAM** field, ensure fields are set as follows:
    - `viewtime` with type `BIGINT`
    - `userid` with type `VARCHAR`
    - `pageid` with type `VARCHAR`



Create a KSQL Stream

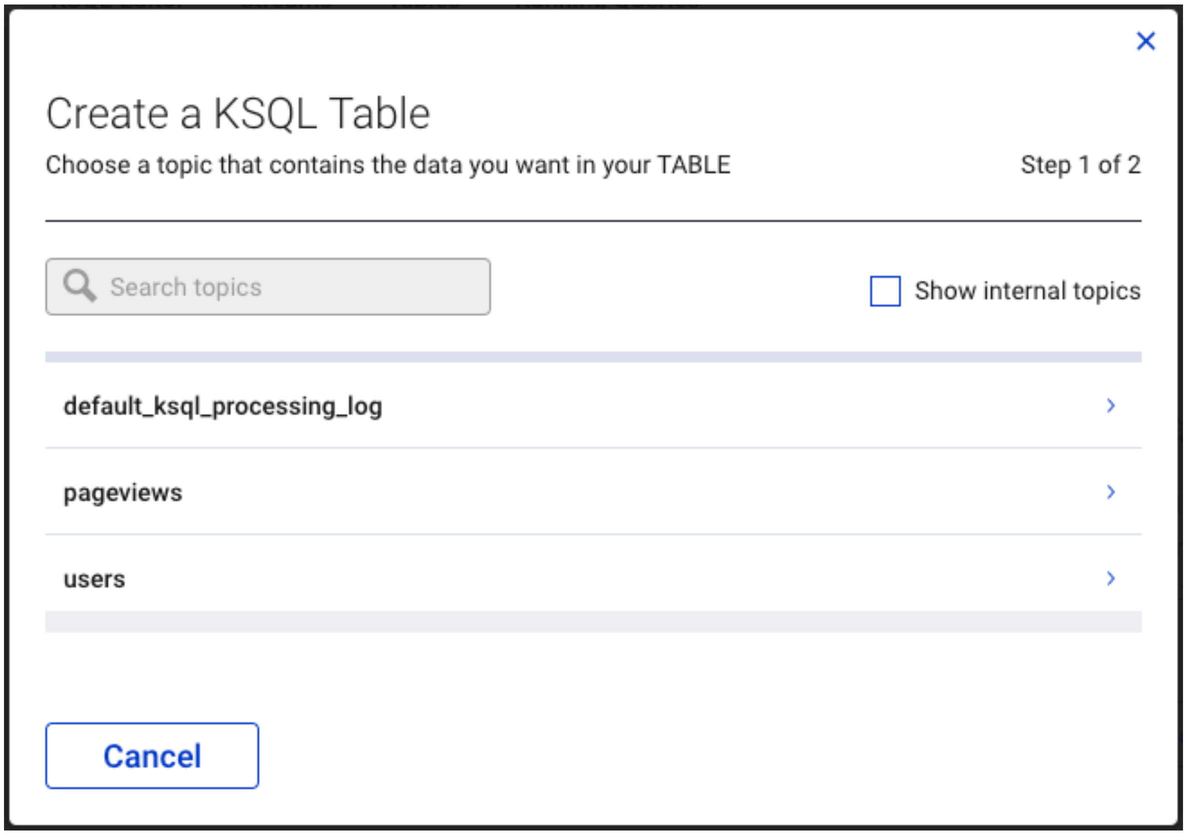
Choose a topic that contains the data you want in your STREAM — Step 2 of 2

KSQL Cluster	KSQL
Topic	pageviews
STREAM name	PAGEVIEWS
Query type	STREAM
Encoding	AVRO
Key	
Timestamp	

Field(s) you'd like to include in your STREAM

Field name	Field type
viewtime	BIGINT
userid	VARCHAR
pageid	VARCHAR

5. Click **Save STREAM**.
6. Click the **Tables** tab -> **Add a Table** and select the `users` topic.


Create a KSQL Table

Choose a topic that contains the data you want in your TABLE

Step 1 of 2

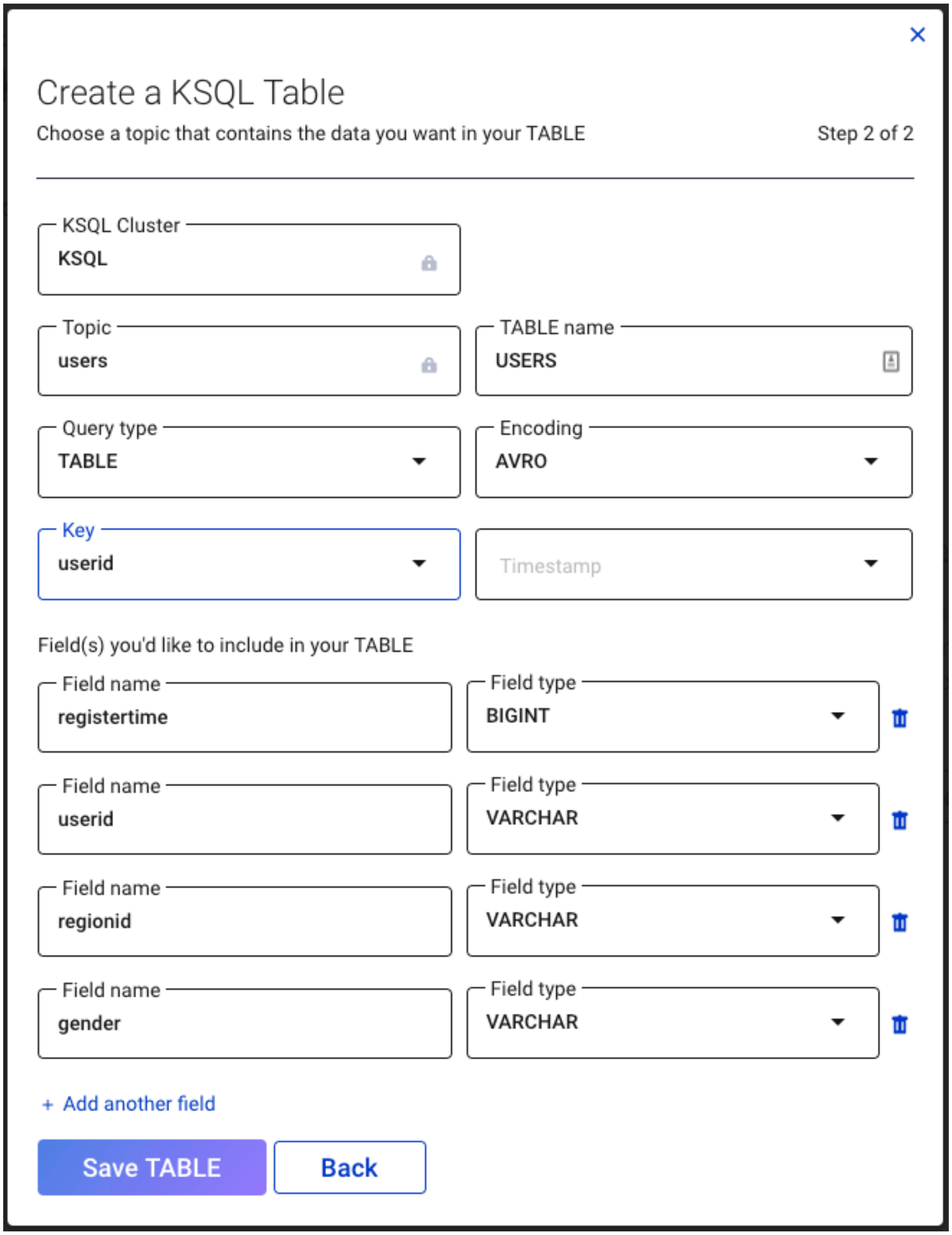
default_ksql_processing_log

pageviews

users

7. Choose your table options:

- In the **Encoding** field, select `AVRO`.
- In the **Key** field, select `userid`.
- In the **Field(s) you'd like to include in your TABLE** field, ensure fields are set as follows:
  - `registertime` with type `BIGINT`
  - `userid` with type `VARCHAR`
  - `regionid` with type `VARCHAR`
  - `gender` with type `VARCHAR`


Create a KSQL Table

Choose a topic that contains the data you want in your TABLE Step 2 of 2

KSQL Cluster: KSQL

Topic: users

TABLE name: USERS

Query type: TABLE

Encoding: AVRO

Key: userid

Timestamp

Field(s) you'd like to include in your TABLE

Field name	Field type
registertime	BIGINT
userid	VARCHAR
regionid	VARCHAR
gender	VARCHAR

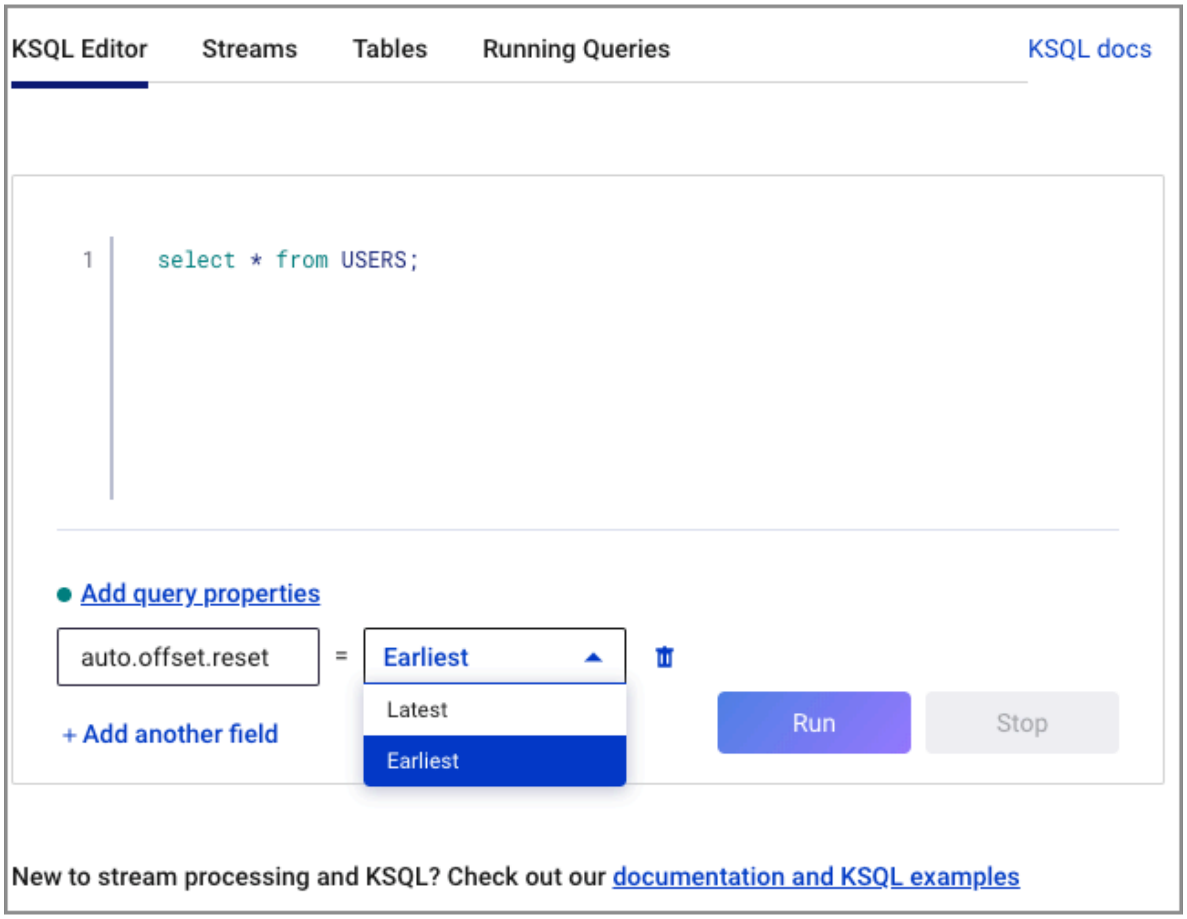
8. Click **Save TABLE**.

Write Queries

These examples write queries using the **KSQL** tab in Control Center.

1. From your cluster, click **KSQL** and choose the **KSQL EDITOR** page.
2. Click **Add query properties** to add a custom query property. Set the `auto.offset.reset` parameter to `earliest`.

This instructs KSQL queries to read all available topic data from the beginning. This configuration is used for each subsequent query. For more information, see the KSQL Configuration Parameter Reference.



```
select * from USERS;
```

auto.offset.reset = Earliest

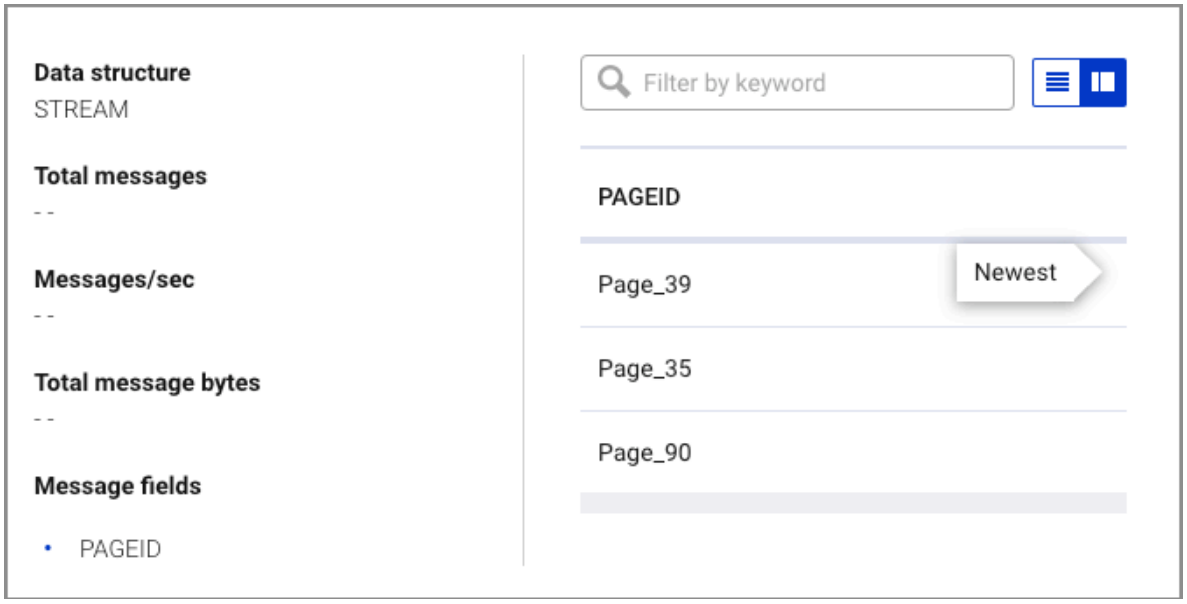
New to stream processing and KSQL? Check out our documentation and KSQL examples

3. Run the following queries.

   1. Create a non-persistent query that returns data from a stream with the results limited to a maximum of three rows.

```
2.  SELECT pageid FROM pageviews LIMIT 3;
```

   Your output should resemble:



Data structure
STREAM

Total messages
- -

Messages/sec
- -

Total message bytes
- -

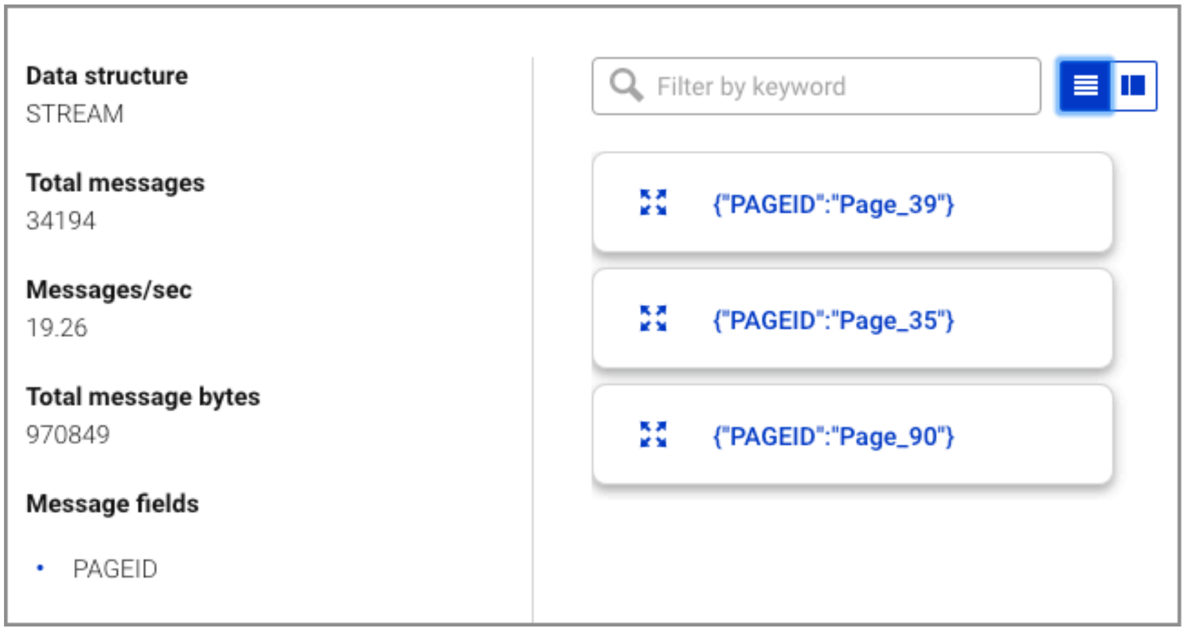
Message fields

- PAGEID

PAGEID
Page_39
Page_35
Page_90

Tip

Click the **Card view** or **Tabular view** icon to change the layout. Click the expand icon to expand a message.



Data structure
STREAM

Total messages
34194

Messages/sec
19.26

Total message bytes
970849

Message fields

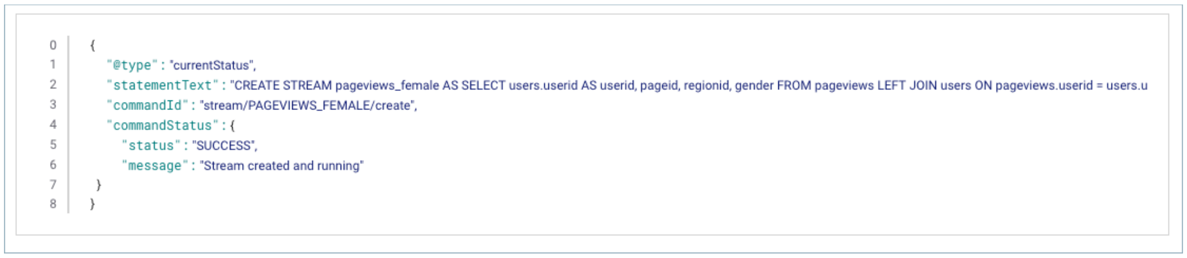
- PAGEID

```
{"PAGEID":"Page_39"}
{"PAGEID":"Page_35"}
{"PAGEID":"Page_90"}
```

3. Create a persistent query that filters for female users. The results from this query are written to the Kafka `PAGEVIEWS_FEMALE` topic. This query enriches the `pageviews` STREAM by doing a `LEFT JOIN` with the `users` TABLE on the user ID, where a condition (`gender = 'FEMALE'`) is met.

```
4.  CREATE STREAM pageviews_female AS SELECT users.userid AS userid, pageid, regionid, gender FROM pagevi
    ews LEFT JOIN users ON pageviews.userid = users.userid WHERE gender = 'FEMALE';
```

Your output should resemble:

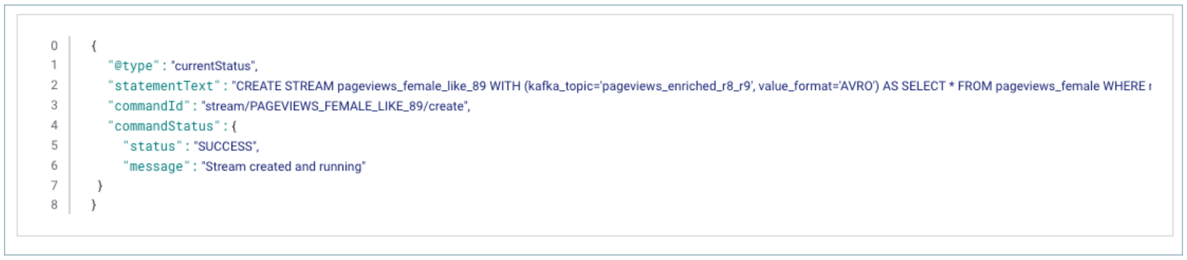


```
{
  "@type" : "currentStatus",
  "statementText" : "CREATE STREAM pageviews_female AS SELECT users.userid AS userid, pageid, regionid, gender FROM pageviews LEFT JOIN users ON pageviews.userid = users.u
  "commandId" : "stream/PAGEVIEWS_FEMALE/create",
  "commandStatus" : {
    "status" : "SUCCESS",
    "message" : "Stream created and running"
  }
}
```

5. Create a persistent query where a condition (`regionid`) is met, using `LIKE`. Results from this query are written to a Kafka topic named `pageviews_enriched_r8_r9`.

```
6. CREATE STREAM pageviews_female_like_89 WITH (kafka_topic='pageviews_enriched_r8_r9', value_format='AV
   RO') AS SELECT * FROM pageviews_female WHERE regionid LIKE '%_8' OR regionid LIKE '%_9';
```

Your output should resemble:

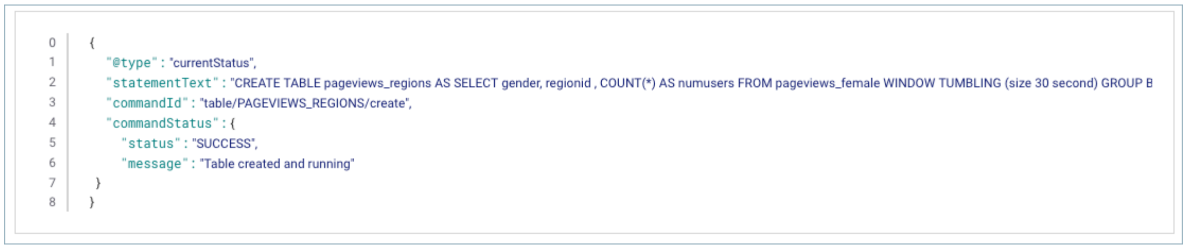


```
{
  "@type" : "currentStatus",
  "statementText" : "CREATE STREAM pageviews_female_like_89 WITH (kafka_topic='pageviews_enriched_r8_r9', value_format='AVRO') AS SELECT * FROM pageviews_female WHERE r
  "commandId" : "stream/PAGEVIEWS_FEMALE_LIKE_89/create",
  "commandStatus" : {
    "status" : "SUCCESS",
    "message" : "Stream created and running"
  }
}
```

7. Create a persistent query that counts the pageviews for each region and gender combination in a tumbling window of 30 seconds when the count is greater than 1. Because the procedure is grouping and counting, the result is now a table, rather than a stream. Results from this query are written to a Kafka topic called `PAGEVIEWS_REGIONS`.

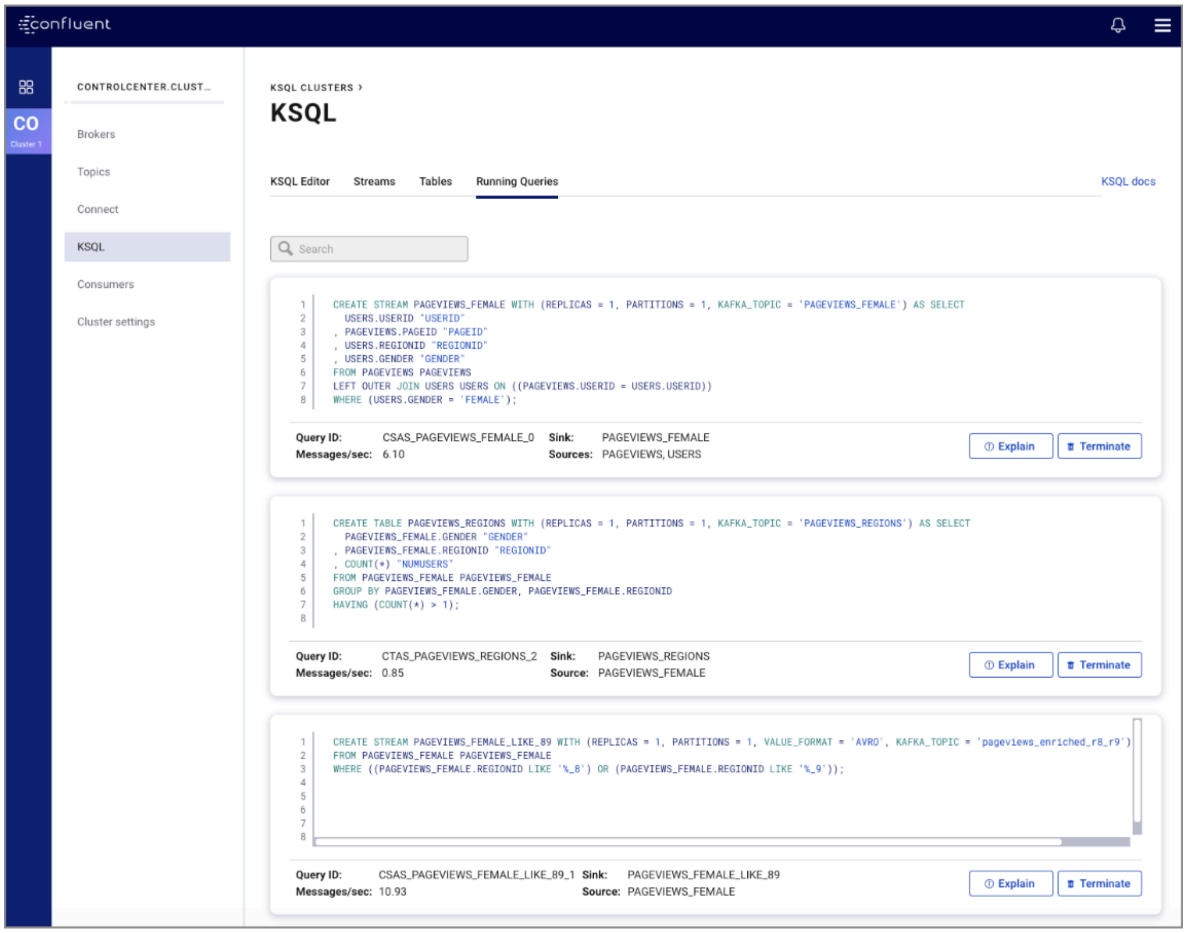
```
8. CREATE TABLE pageviews_regions AS SELECT gender, regionid , COUNT(*) AS numusers FROM pageviews_femal
   e WINDOW TUMBLING (size 30 second) GROUP BY gender, regionid HAVING COUNT(*) > 1;
```

Your output should resemble:



```
{
  "@type" : "currentStatus",
  "statementText" : "CREATE TABLE pageviews_regions AS SELECT gender, regionid , COUNT(*) AS numusers FROM pageviews_female WINDOW TUMBLING (size 30 second) GROUP B
  "commandId" : "table/PAGEVIEWS_REGIONS/create",
  "commandStatus" : {
    "status" : "SUCCESS",
    "message" : "Table created and running"
  }
}
```

9. Click **RUNNING QUERIES**. You should see the following persisted queries:



```
CREATE STREAM PAGEVIEWS_FEMALE WITH (REPLICAS = 1, PARTITIONS = 1, KAFKA_TOPIC = 'PAGEVIEWS_FEMALE') AS SELECT
  USERS.USERID "USERID"
, PAGEVIEWS.PAGEID "PAGEID"
, USERS.REGIONID "REGIONID"
, USERS.GENDER "GENDER"
FROM PAGEVIEWS PAGEVIEWS
LEFT OUTER JOIN USERS USERS ON ((PAGEVIEWS.USERID = USERS.USERID))
WHERE (USERS.GENDER = 'FEMALE');
```

Query ID: CSAS_PAGEVIEWS_FEMALE_0 **Sink:** PAGEVIEWS_FEMALE
Messages/sec: 6.10 **Sources:** PAGEVIEWS, USERS

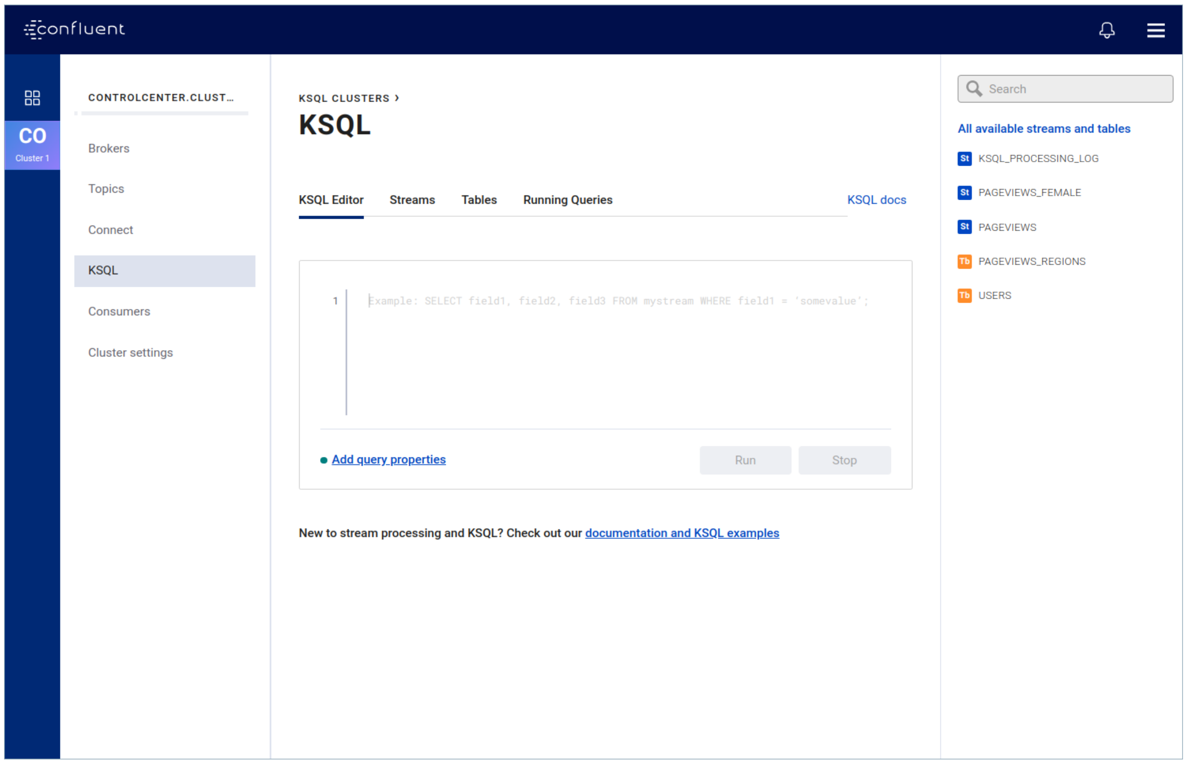
```
CREATE TABLE PAGEVIEWS_REGIONS WITH (REPLICAS = 1, PARTITIONS = 1, KAFKA_TOPIC = 'PAGEVIEWS_REGIONS') AS SELECT
  PAGEVIEWS_FEMALE.GENDER "GENDER"
, PAGEVIEWS_FEMALE.REGIONID "REGIONID"
, COUNT(*) "NUMUSERS"
FROM PAGEVIEWS_FEMALE PAGEVIEWS_FEMALE
GROUP BY PAGEVIEWS_FEMALE.GENDER, PAGEVIEWS_FEMALE.REGIONID
HAVING (COUNT(*) > 1);
```

Query ID: CTAS_PAGEVIEWS_REGIONS_2 **Sink:** PAGEVIEWS_REGIONS
Messages/sec: 0.85 **Source:** PAGEVIEWS_FEMALE

```
CREATE STREAM PAGEVIEWS_FEMALE_LIKE_89 WITH (REPLICAS = 1, PARTITIONS = 1, VALUE_FORMAT = 'AVRO', KAFKA_TOPIC = 'pageviews_enriched_r8_r9')
FROM PAGEVIEWS_FEMALE PAGEVIEWS_FEMALE
WHERE ((PAGEVIEWS_FEMALE.REGIONID LIKE '%_8') OR (PAGEVIEWS_FEMALE.REGIONID LIKE '%_9'));
```

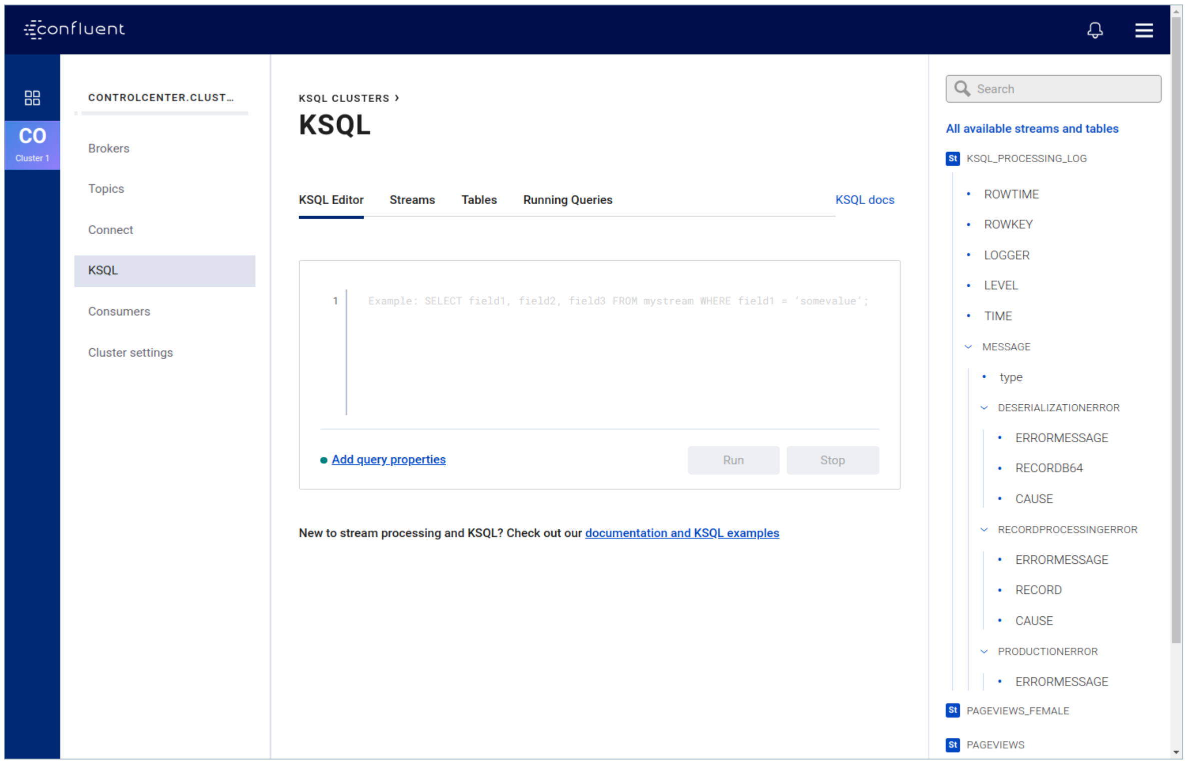
Query ID: CSAS_PAGEVIEWS_FEMALE_LIKE_89_1 **Sink:** PAGEVIEWS_FEMALE_LIKE_89
Messages/sec: 10.93 **Source:** PAGEVIEWS_FEMALE

10. Click **KSQL Editor**. On the right side of the page, find the **All available streams and tables** pane, which shows all of the streams and tables that you can access.



New to stream processing and KSQL? Check out our documentation and KSQL examples

11. In the **All available streams and tables** section, click **KSQL_PROCESSING_LOG** to view the stream's schema, including nested data structures.



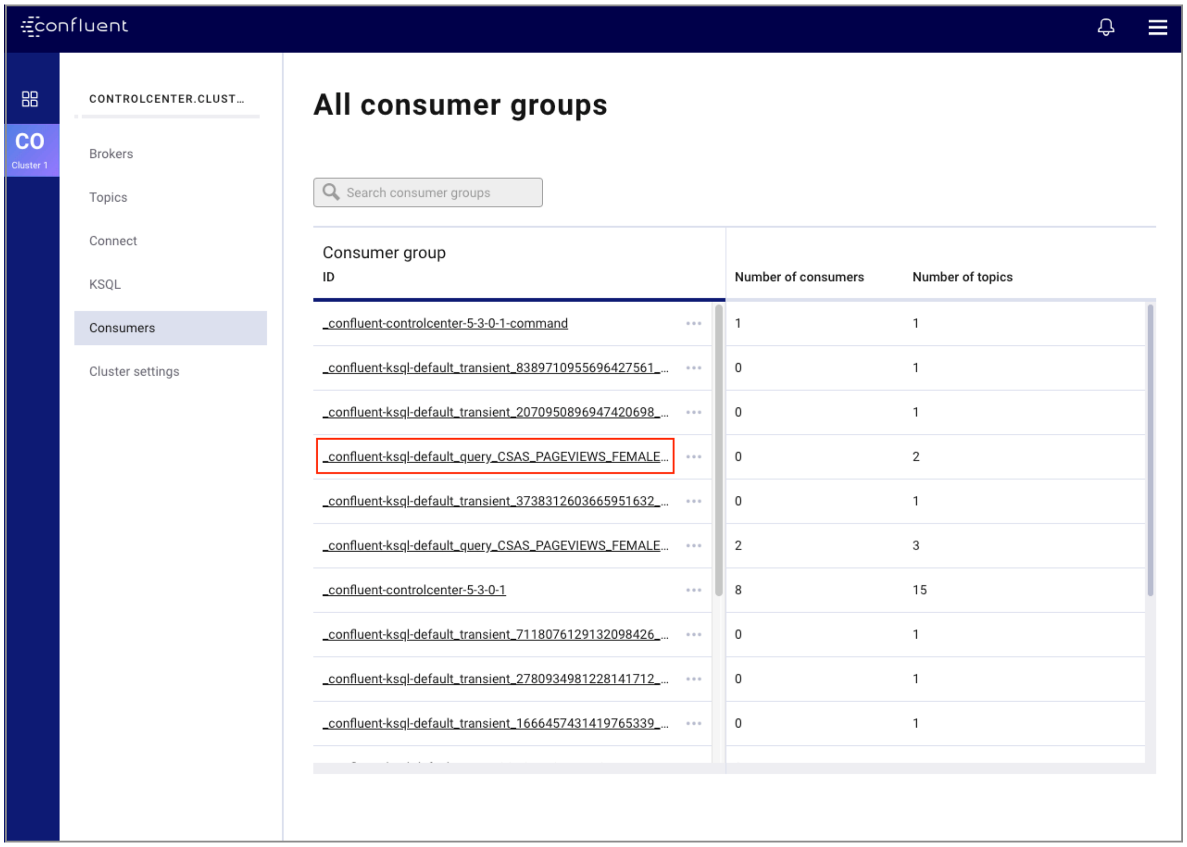
```
Example: SELECT field1, field2, field3 FROM mystream WHERE field1 = 'somevalue';
```

New to stream processing and KSQL? Check out our documentation and KSQL examples

View Your Stream in Control Center

Navigate to the **Consumers** tab at http://localhost:9021/monitoring/consumer/lag/consumerGroups to view the consumers created by KSQL.

Click the consumer group ID to view details for the `_confluent-ksql-default_query_CSAS_PAGEVIEWS_FEMALE_0` consumer group.

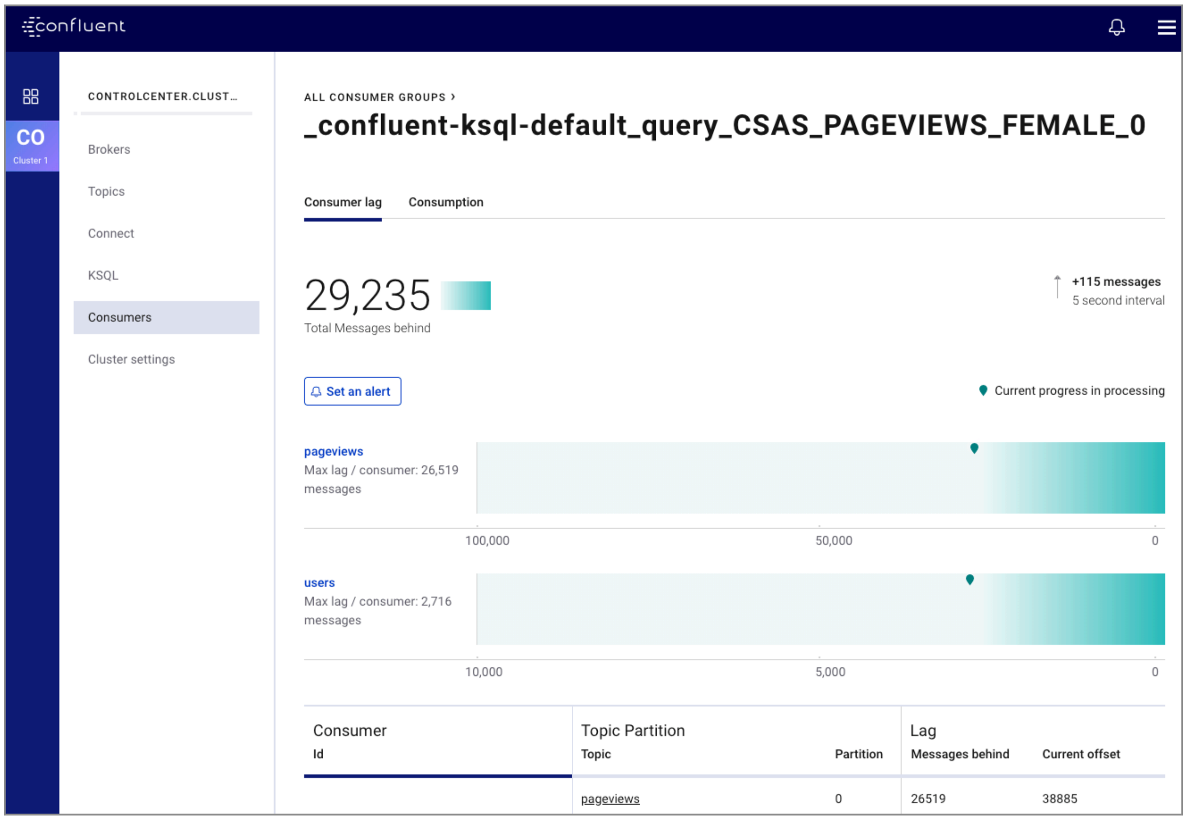


All consumer groups

Search consumer groups

Consumer group ID	Number of consumers	Number of topics
_confluent-controlcenter-5-3-0-1-command	1	1
_confluent-ksql-default_transient_8389710955696427561_...	0	1
_confluent-ksql-default_transient_2070950896947420698_...	0	1
_confluent-ksql-default_query_CSAS_PAGEVIEWS_FEMALE...	0	2
_confluent-ksql-default_transient_3738312603665951632_...	0	1
_confluent-ksql-default_query_CSAS_PAGEVIEWS_FEMALE...	2	3
_confluent-controlcenter-5-3-0-1	8	15
_confluent-ksql-default_transient_7118076129132098426_...	0	1
_confluent-ksql-default_transient_2780934981228141712_...	0	1
_confluent-ksql-default_transient_1666457431419765339_...	0	1

From this page you can see the consumer lag and consumption values for your streaming query.



_confluent-ksql-default_query_CSAS_PAGEVIEWS_FEMALE_0

29,235 Total Messages behind

+115 messages 5 second interval

pageviews Max lag / consumer: 26,519 messages

users Max lag / consumer: 2,716 messages

Consumer	Topic Partition		Lag	
Id	Topic	Partition	Messages behind	Current offset
	pageviews	0	26519	38885

For more information, see Consumers.

Stop Docker

When you are done working with Docker, you can stop and remove Docker containers and images.

1. View a list of all Docker container IDs.

```
2. docker container ls -aq
```

3. Run the following command to stop the Docker containers for Confluent:

```
4. docker container stop $(docker container ls -a -q -f "label=io.confluent.docker")
```

5. Run the following commands to stop the containers and prune the Docker system. Running these commands deletes containers, networks, volumes, and images; freeing up disk space:

```
6. docker container stop $(docker container ls -a -q -f "label=io.confluent.docker") && docker system prune
   -a -f --volumes
```

Tip

Remove the filter label for Confluent Docker (`-f "label=io.confluent.docker"`) to clear all Docker containers from your system.

You can rebuild and restart the containers at any time using the `docker-compose up -d --build` command.

For more information, refer to the official Docker documentation.