

Workshop

How to build streaming applications with Confluent Cloud (step-by-step guide)

Version 23/May/2023

Table of Contents

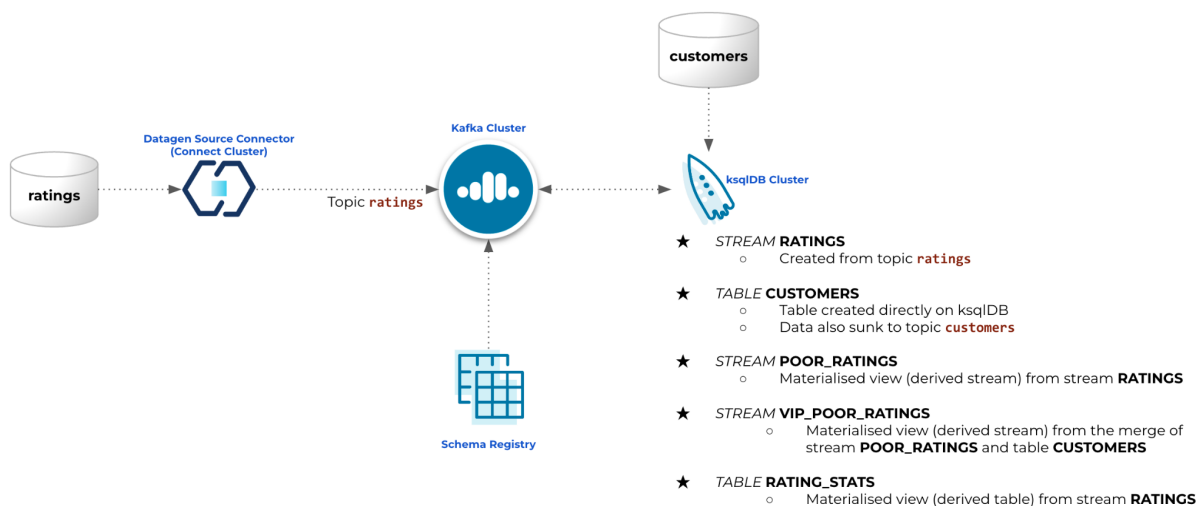
1. Introduction	2
1.1 Demo diagram	2
2. Pre-reqs / creating the resources	2
2.1 Register for Confluent Cloud	2
2.2 Create an Environment and a Schema Registry Cluster	3
2.3 Create a Kafka Cluster	4
2.4 Add an API Key	6
2.5 Create a ksqlDB Cluster	6
2.6 Create a new Topic	8
2.7 Create a Connect Cluster	9
3. ksqlDB	11
3.1 Syntax Reference	11
3.2 Accessing the cluster	12
3.3 See available Kafka topics and data	12
4. Getting started with DDL	13
4.1 Create the Ratings data stream	13
4.2 Create a new stream with Customer data	15
4.3 Identify the unhappy customers	18
4.4 Using tables as a Materialised Cache	19
4.5 Monitoring our Queries	20
4.5.1 View Consumer Lag for a Query	21
5. Extra credit	21
6. Follow-on discussion points	22
6.1 Session properties	22
7. Further resources	22
8. Deleting the resources created	22
8.1 Deleting the ksqlDB cluster	23
8.2 Deleting the Connect cluster	23
8.3 Deleting the Kafka cluster	23
8.4 Deleting the Environment	23

1. Introduction

[ksqlDB](#) is the streaming SQL engine for Apache Kafka. This workshop will step through some practical examples of how to use [Confluent Cloud](#) and ksqlDB to build powerful real-time stream-processing applications:

- Quickly build your environment and clusters using Confluent Cloud
- [Datagen source connector](#) to generate dummy/test data
- ksqlDB to filter streams of data and join live streams of events with reference data (e.g. from a database)
- Continuous, stateful aggregations

1.1 Demo diagram



2. Pre-reqs / creating the resources

First things first, let's sign up to Confluent Cloud to create the environment, clusters and make sure we have access to everything we need.

2.1 Register for Confluent Cloud

Note: If you already have a Confluent Cloud account, you can skip ahead to [Create an Environment and a Schema Registry Cluster](#).

1. Head over to the Confluent Cloud signup page (<https://confluent.cloud/signup>) and enter your name, email address, and password (New signups receive USD 400 to spend during their first 30 days. No payment method required)

2. Click the **Start Free** button. (Make sure to keep track of your password, as you'll need it to log in to Confluent Cloud later on)

☐ Email me about products, services, & events from Confluent

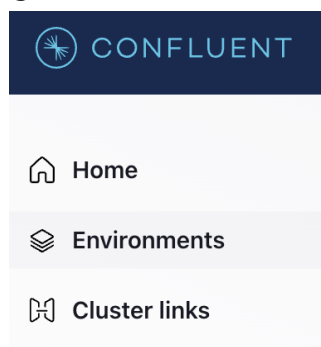
START FREE

3. Watch your inbox for a confirmation email. Once you get the email, follow the link to proceed
4. At this point, you will be asked to create a cluster

2.2 Create an Environment and a Schema Registry Cluster

Now that you have Confluent Cloud set up, we are going to create an environment specifically for this workshop. This will ensure your environment is isolated from any other work you might be doing, and it will make cleanup easier.

1. From the left-hand navigation menu select **Environments**



2. Click **+ Add cloud environment**


[+ Add cloud environment](#)

3. Name your environment **workshop-ksqldb**

- When offered a choice on which [Stream Governance package](#) to choose, click **Begin Configuration** under the **Essentials** option

Stream Governance Packages

Confluent's Stream Governance suite establishes trust in the data streams moving throughout your cloud environments and delivers an easy, self-service experience for more teams to discover, understand, and put streaming data to work.


Essentials

The fundamentals for getting started.

Stream Quality
Schema Registry & validation

- 99.5% uptime SLA
- 1,000 schemas included*
- 9 cloud regions supported


Stream Catalog
Data organization & discoverability

- Auto-technical metadata ingestion
- Tags metadata
- Cloud UI & REST API

Stream Lineage
Data origin & tracking

- Real-time data streams lineage

*Starting at \$0.002/schema/hour after 1,000 schemas per environment


Advanced

Enterprise-ready controls for data in motion.

Stream Quality
Schema Registry & validation

- 99.95% uptime SLA
- 20,000 schemas included
- 30 cloud regions supported

Stream Catalog
Data organization & discoverability

- All existing Essentials features +
 - Business metadata
 - GraphQL API

Stream Lineage
Data origin & tracking

- All existing Essentials features +
 - Point-in-time lineage
 - Lineage search


- Select which cloud provider (AWS, GCP or Azure) and region you want to create your [Schema Registry](#) and [Stream Catalog](#) in (i.e. where you will be storing the metadata), then click on **Enable** button

2.3 Create a Kafka Cluster

Next, we need to create a Kafka Cluster for the workshop.

- Inside the **workshop-ksqldb** environment click **Create cluster on my own**. You'll be given a choice of what kind of cluster to create


Create cluster
 1. Select cluster type ○ ○ ○ ○ ○


Basic

For learning and exploring Kafka and Confluent Cloud.

Ingress	up to 250 MB/s
Egress	up to 750 MB/s
Storage	up to 5,000 GB
Client connections	up to 1,000
Partitions	up to 4,096 (includes 10 free partitions)
Uptime SLA	up to 99.5%


Begin configuration


Standard

For production-ready use cases. Full feature set and standard limits.

Ingress	up to 250 MB/s
Egress	up to 750 MB/s
Storage	unlimited
Client connections	up to 1,000
Partitions	up to 4,096 (includes 500 free partitions)
Uptime SLA	up to 99.99% ⓘ

Begin configuration


Dedicated

For use cases with high traffic or that require private networking.

Price as sized: 1 CKU

Ingress	up to 50 MB/s
Egress	up to 150 MB/s
Storage	unlimited
Client connections	up to 9,000
Partitions	up to 4,500
Uptime SLA	up to 99.99% ⓘ

Begin configuration

- Basic clusters used in the context of this workshop won't incur much cost, and the amount of free usage that you received will be more than enough to cover it and even allow you to continue playing with it for the next 30 days (Your free allowance is for USD 400 or 30 days after signing up, whichever comes first)

- On the next page, choose your cloud provider (AWS, GCP or Azure), region, and availability (zone), then select **Continue** (Costs will vary with these choices, but they are clearly shown on the dropdown, so you'll know what you're getting)

Create cluster

1. Select cluster type — 2. Region/zones — 3. Review and launch



 Google Cloud

 Microsoft Azure

Region*
 Iowa (us-central1)

Availability* ⓘ
 Single zone

[Go back](#)

\$0.00 /hr + usage

[Continue](#)

- On the next screen, give your Kafka cluster a name, and select **Launch cluster**

Create cluster

1. Select cluster type — 2. Region/zones — 3. Review and launch

Cluster name ⓘ
 cluster_0

Base cost	\$0 /hr
Write	\$0.11 /GB
Read	\$0.11 /GB
Storage	\$0.00013889 /GB-hour
Partitions	\$0.004 /Partition-hour (includes 10 free partitions)

Configuration & cost Usage limits Uptime SLA

Cluster configuration

ⓘ Settings marked with an asterisk (*) cannot be changed once you launch your cluster

Cluster type	Basic	*Provider	Google Cloud Platform
*Region	us-central1	*Networking	Internet

[Go back](#)

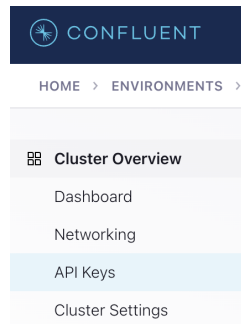
[Review payment method](#)

[Launch cluster](#)

2.4 Add an API Key

We will need an API Key to allow applications to access our Kafka cluster, so let's create one.

1. From the left-hand navigation in your cluster, navigate to **Cluster Overview** > **API keys**



2. Click **+ Add key**



3. Create a key with Global access.

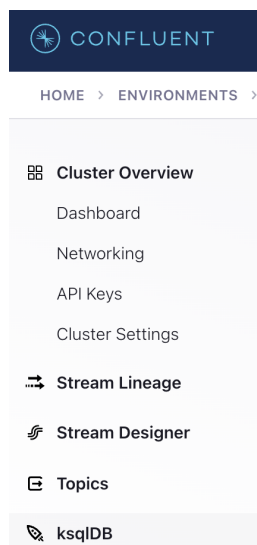
Note: For secure production environments, you would want to select Granular access and configure it more securely.

4. Give a description to your API key, then click **Download and continue**. Make sure to save the key somewhere for future use.

2.5 Create a ksqlDB Cluster

Next, we need to create a ksqlDB Cluster for the workshop.

1. From the left-hand navigation in your Kafka cluster, navigate to **ksqlDB**



2. Click **Create cluster myself**

ksqlDB

Stream Processing with ksqlDB


Stream processing involves ingesting a continuous data stream to analyze, filter, transform, or enhance data in real time. ksqlDB makes it easy by allowing you to enrich your real-time data streams in-flight with familiar SQL syntax. See [how ksqlDB works](#).

[Start with tutorial](#)
[Create cluster myself](#)

3. Select **Use an existing API key**

New cluster


1. Access control ○ ○ ○ ○ ○



Global access

Allow your ksqlDB cluster to access everything you can access. Cluster access will be linked to your account

**Recommended for development.*



Granular access

Limit the access for your cluster. Manage cluster access through a service account.

**Recommended for production.*

Note: For secure production environments, you would want to select Granular access and configure it more securely.

4. On the next screen, give your ksqlDB cluster a name, set cluster size to **1** (one), select **Default** on Configuration options and click **Launch cluster!**

New cluster

1. Access control — 2. Name and sizing

Cluster name*
ksqlDB_cluster_0

Cluster size*
1 \$0.22222 USD /hr

Confluent streaming units (CSUs) determine the amount of data your cluster can process. High Availability is enabled by default on clusters with 8 CSUs or more.
[Learn more](#)

Configuration options

Default

Advanced

[Back](#)

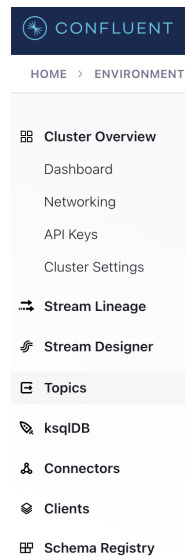
Hourly cost: \$0.22222
Estimated monthly cost: \$160.00

Launch cluster!

5. It should take around 5 to 10 minutes to have your ksqlDB cluster provisioned, so let's head over to the next section

2.6 Create a new Topic

1. From the left-hand navigation in your Kafka cluster, navigate to **Topics**



2. Click **+ Add topic**



3. Name the topic **ratings** and ensure that the **Number of partitions** is set to **6** (six), then click **Create with defaults**

WORKSHOP-KSQLDB > CLUSTER_0 > TOPICS >

New topic

Topic name* ⓘ
 ratings

Partitions* ⓘ
 6

[Show advanced settings](#)
[Cancel](#)
[Create with defaults](#)

4. On the next screen, **skip** the creation of the schema (the connector we will create in the next section will do that automatically)

Your topic has been successfully created!

Define a data contract

Create a schema to describe the data in your topic and ensure producers and consumers have a contract on how to communicate. Enforce data integrity when your clients use a schema to produce messages to the topic and maintain data compatibility as you evolve the schema over time. Plus, Confluent automatically catalogs your schemas, so you can tag and search all schema fields. [Learn More](#)

☒ Create a schema for message values Most common
☐ Create a schema for message keys

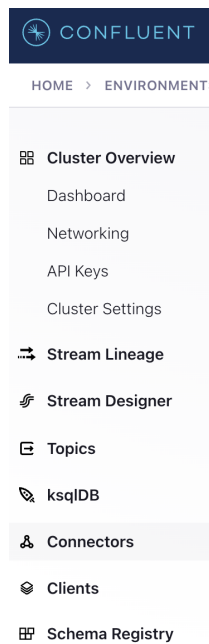
[Skip](#)
[Create Schema](#)

2.7 Create a Connect Cluster

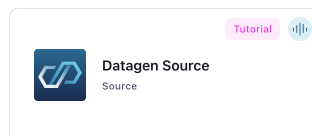
Next, we need to create a Connect Cluster for the workshop. That cluster will generate dummy customer rating data into Kafka.

In reality, our Kafka topic would probably be populated from an application using the producer API to write messages to it. Here, we're going to use a data generator that's available as a connector for Kafka Connect.

1. From the left-hand navigation in your Kafka cluster, navigate to **Connectors**



2. Select the **Datagen Source** connector



3. On the **Topic selection** page, select the topic **ratings** created in the previous section, click **Continue**

Add Datagen Source connector

1. Topic selection 2. Kafka credentials 3. Configuration 4. Sizing 5. Review and launch

Select or create new topics

Choose which topics you want to connect

(1) topic selected
+ Add new topic


	Topics	Partitions	Throughput	
	Topic name	Total partitions	Bytes/sec produced	Bytes/sec consumed
<input type="radio"/>	pksqlc-p1j6y-processing-log	8	--	--
<input checked="" type="radio"/>	ratings	6	--	--

[Go back](#)
Continue

- On the **Kafka credentials** page, select **Use an existing API key** (created previously), enter the credentials on the **Enter API key and Secret** and click **Continue**


Add Datagen Source connector

1. Topic selection — 2. Kafka credentials — 3. Configuration — 4. Sizing — 5. Review and launch



Global access


Allow your connector to access everything you have access to. Connector access will be linked to your account.



Granular access

Limit the access for your connector. Manage connector access through a service account.

*Recommended for production.



Use an existing API key

Enter an API key and secret pair that you have stored.

Enter API Key and Secret

Kafka API Key*

Kafka API Secret*

[Go back](#) [Continue](#)

- On the **Configurations** page, select **AVRO** (Select output record value format), select **Ratings** (Select a template) and click **Continue**

Add Datagen Source connector

1. Topic selection — 2. Kafka credentials — 3. Configuration — 4. Sizing — 5. Review and launch

Select output record value format

AVRO

JSON

JSON_SR

PROTOBUF

Select a template

Orders

Users

Clickstream

Pageviews

Product

Ratings

Stock trades

Inventory

[Go back](#) [Request preview](#) [Continue](#)

- On the **Sizing** page, keep the default settings (1 task) and click **Continue**

Add Datagen Source connector

1. Topic selection — 2. Kafka credentials — 3. Configuration — 4. Sizing — 5. Review and launch

Topic summary

Topics selected 1

Partitions selected from topics 6

Connector sizing

Minimum number of tasks 1 (recommended)

Connector sizing: 1

1 task / \$0.03472222/hr

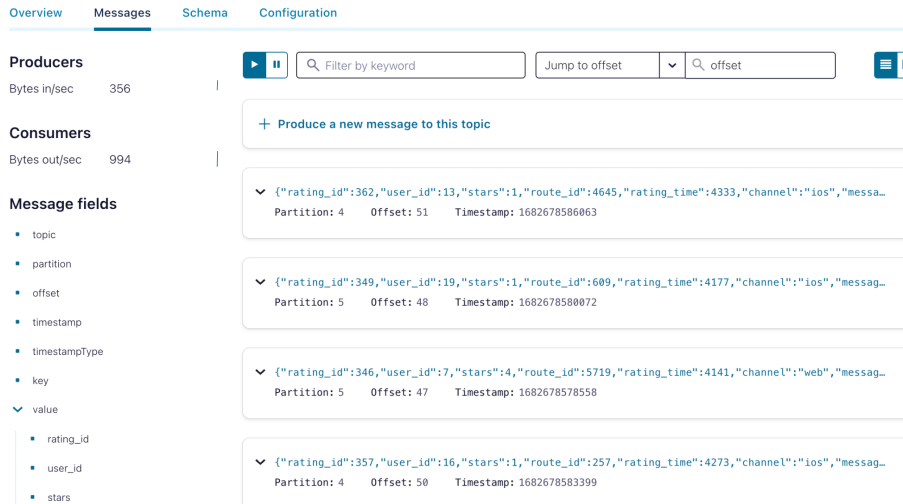
Tasks can be scaled up at a later time for additional throughput capacity.

[Go back](#) [Request preview](#) [Continue](#)

- On the **Review and launch** page click **Continue**

8. The connector is being provisioned and should take couple of minutes to complete
9. Once the connector is created, navigate to **Topics** (from the left-hand navigation in your Kafka cluster), select the topic **ratings**, click tab **Messages** and see the messages coming through

ratings



After the connector cluster is deployed and start producing data to the topic **ratings**, it will set the AVRO schema as follows:

```
{
  "namespace": "ksql",
  "name": "ratings",
  "type": "record",
  "fields": [
    {"name": "rating_id", "type": "long"},
    {"name": "user_id", "type": "int"},
    {"name": "stars", "type": "int"},
    {"name": "route_id", "type": "int"},
    {"name": "rating_time", "type": "long"},
    {"name": "channel", "type": "string"},
    {"name": "message", "type": "string"}
  ]
}
```

3. ksqlDB

3.1 Syntax Reference

You will find it helpful to keep a copy of the KSQL syntax guide open in another browser tab: <https://docs.ksqldb.io/en/latest/reference/>

3.2 Accessing the cluster

ksqlDB can be accessed via either the command line interface (CLI), Confluent Cloud, or the documented REST API

(<https://docs.ksqldb.io/en/latest/developer-guide/api>).

In this workshop we will mainly be using Confluent Cloud.

1. From the left-hand navigation in your Kafka cluster, navigate to **ksqlDB**
2. Select the ksqlDB cluster created previously

3.3 See available Kafka topics and data

Remember we discussed in the presentation that KSQL works with Streams and Tables, and these are just abstractions for working with data in topics ? So the first thing we will do is find what topics we have available to work with on our Kafka cluster.

1. On the **Editor** tab enter the command **show topics;** and click **Run query:**

⚠ You have no running queries yet
▼

Editor
Flow
Streams
Tables
Persistent queries
Performance
Settings
CLI instructions

```
1 show topics;
```

• [Add query properties](#)

auto.offset.reset

=

Latest

⊞

[+Add another field](#)

Stop

Run query

```

1 {
2   "etype": "kafka_topics",
3   "statementText": "show topics;",
4   "topics": [
5     {
6       "name": "pksqlc-plj6y-processing-log",
7       "replicaInfo": [
8         3,
9         3,
10        3,
11        3,
12        3,
13        3,
14        3,
15        3
16      ]
17    },
18    {
19      "name": "ratings",
20      "replicaInfo": [
21        3,
22        3,
23        3,
24        3,
25        3
26      ]
27    }
28  ],
29  "warnings": [
30  ]
31  }
32 }
33 
```

The list of topics you see here is exactly the same as you would see in the **Topics** section, except that here we are getting the list via a KSQL command instead of browsing graphically for it.

We can also investigate some data from those topics before working with them:

2. Still on the **Editor** tab enter the command **below** and click **Run query**:

```
print 'ratings' from beginning limit 3;
```

The event stream driving this example (the data in the **ratings** topic) is a simulated stream of events representing the ratings left by users on a mobile app or website, with fields including the device type that they used, the star rating (a score from 1 to 5), and an optional comment associated with the rating.

Notice that we don't need to know the format of the data when printing a topic. ksqlDB introspects the data based on its schema (AVRO in this case) and understands how to deserialize it.



From now on, when requesting to issue a given KSQL command, go to the **Editor** tab, enter the command and click **Run query** (always make sure the command ends with a semicolon ';').

To stop a running query click **Stop Query**.

4. Getting started with DDL

To make use of our ratings and customers topics in KSQL we first need to define some Streams and/or Tables over them. You can check what streams or tables you have at any time:

```
show streams;
show tables;
```

4.1 Create the Ratings data stream

Register the RATINGS data as a KSQL stream, sourced from the **ratings** topic. Make sure to select the query property **auto.offset.reset** to **Earliest**.

```
create stream ratings with (
  kafka_topic='ratings',
  value_format='avro'
);
```

Note: Since the stream will be directly derived from a Kafka topic, there is no need to define/declare the field/column names as ksqlDB will pick it up from the AVRO schema (it would also work if the schema were Protobuf or JSON).

Editor
Flow
Streams
Tables
Persistent queries
Performance
Settings
CLI instructions

```

1 create stream ratings with (kafka_topic='ratings', value_format='avro');

```

• Add query properties

auto.offset.reset = Earliest

+Add another field

Stop Run query

```

1 {
2   "@type": "currentStatus",
3   "statementText": "CREATE STREAM RATINGS (RATING_ID BIGINT, USER_ID INTEGER, STARS INTEGER, ROUTE_ID INTEGER, RATING_TIME BIGINT,
4   CHANNEL STRING, MESSAGE STRING) WITH (CLEANUP_POLICY='delete', KAFKA_TOPIC='ratings', KEY_FORMAT='KAFKA', VALUE_FORMAT='AVRO');",
5   "commandId": "stream/'RATINGS'/create",
6   "commandStatus": {
7     "status": "SUCCESS",
8     "message": "Stream created",
9     "queryId": null
10  },
11  "commandSequenceNumber": 2,
12  "warnings": [
13  ]
14 }

```

Notice that here we are using the Schema Registry with our Avro-formatted data to pull in the schema of this stream automatically. If interested, you can compare the schema which ksqlDB has created here against what was registered in the Schema Registry in Confluent Cloud (they will be the same!) If our data were in some other format which can't be described in the Schema Registry, such as CSV messages, then we would also need to specify each column and its datatype in the create statement.

Check your creation with **describe ratings;** and then run a couple of select queries. For example:

```

select * from ratings
where channel like 'iOS%'
emit changes;

```

What happens? Why?

The **emit changes** at the end of your select query is basically telling ksqlDB to keep following the topic which underlies your stream. As new records are written into that topic, ksqlDB will read them, filter them through the where clause, and write back to your terminal any events which pass the specified condition. In other words, this query will never end! This is one of the key differences about a *streaming* database.

You can also try `describe ratings extended;` to see more technical information about the data flowing through your new stream.

4.2 Create a new stream with Customer data

We need to create a lookup table for Customer data, for that we will use ksqlDB and insert the records directly into it, just like in a transactional SQL/NOSQL database. First let's create the table:

```
create table customers (  
    user_id INTEGER PRIMARY KEY,  
    first_name STRING,  
    last_name STRING,  
    email STRING,  
    gender STRING,  
    club_status STRING,  
    comments STRING  
) with (  
    kafka_topic='customers',  
    value_format='AVRO',  
    partitions=6  
);
```

Quickly query all records to check it:

```
select * from customers EMIT CHANGES;
```

What happens when you query from this new table?

Now, let's populate the table with some customer data (you can run all commands at once, multiple values not yet supported):

```
insert into customers (user_id, first_name, last_name, email, gender,  
club_status, comments) VALUES (-1, 'Joey', 'Nelson', 'nelsonj@kuka.com',  
'Male', 'silver', 'Rock band pop start');
```

```
insert into customers (user_id, first_name, last_name, email, gender,  
club_status, comments) VALUES (0, 'Melissa', 'Armstrong',  
'melarm@yoga.io', 'Female', 'bronze', 'Yoga teacher');
```

```
insert into customers (user_id, first_name, last_name, email, gender,  
club_status, comments) VALUES (1, 'Rica', 'Blaisdell',  
'rblaisdell@rambler.ru', 'Female', 'bronze', 'Universal optimal  
hierarchy');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (2, 'Ruthie', 'Brockherst',
'rbrockherst1@ow.ly', 'Female', 'platinum', 'Reverse-engineered tangible
interface');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (3, 'Mariejeanne', 'Cocci',
'mcocci2@techcrunch.com', 'Female', 'bronze', 'Multi-tiered
bandwidth-monitored capability');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (4, 'Hashim', 'Rumke', 'hrumke3@sohu.com',
'Male', 'platinum', 'Self-enabling 24/7 firmware');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (5, 'Hansiain', 'Coda', 'hcoda4@senate.gov',
'Male', 'platinum', 'Centralised full-range approach');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (6, 'Robinet', 'Leheude',
'rleheude5@reddit.com', 'Female', 'platinum', 'Virtual upward-trending
definition');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (7, 'Fay', 'Huc', 'fhuc6@quantcast.com',
'Female', 'bronze', 'Operative composite capacity');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (8, 'Patti', 'Rosten', 'prosten7@ihg.com',
'Female', 'silver', 'Integrated bandwidth-monitored instruction set');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (9, 'Even', 'Tinham',
'etinham8@facebook.com', 'Male', 'silver', 'Virtual full-range
info-mediaries');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (10, 'Brena', 'Tollerton',
'btollerton9@furl.net', 'Female', 'silver', 'Diverse tangible
methodology');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (11, 'Alexandro', 'Peeke-Vout',
'apeekevouta@freewebs.com', 'Male', 'gold', 'Ameliorated value-added
orchestration');
```



```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (12, 'Sheryl', 'Hackwell',
'shackwellb@paginegialle.it', 'Female', 'gold', 'Self-enabling global
parallelism');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (13, 'Laney', 'Toopin', 'ltoopinc@icio.us',
'Female', 'platinum', 'Phased coherent alliance');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (14, 'Isabelita', 'Talboy',
'italboyd@imageshack.us', 'Female', 'gold', 'Cloned transitional
synergy');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (15, 'Rodrique', 'Silverton',
'rsilvertone@umn.edu', 'Male', 'gold', 'Re-engineered static
application');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (16, 'Clair', 'Vardy',
'cvardyf@reverbnation.com', 'Male', 'bronze', 'Expanded bottom-line
Graphical User Interface');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (17, 'Brianna', 'Paradise',
'bparadiseg@nifty.com', 'Female', 'bronze', 'Open-source global toolset');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (18, 'Waldon', 'Keddey',
'wkeddeyh@weather.com', 'Male', 'gold', 'Business-focused multi-state
functionalities');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (19, 'Josiah', 'Brockett',
'jbrocketti@com.com', 'Male', 'gold', 'Realigned didactic
info-mediaries');
```

```
insert into customers (user_id, first_name, last_name, email, gender,
club_status, comments) VALUES (20, 'Anselma', 'Rook', 'arookj@europa.eu',
'Female', 'gold', 'Cross-group 24/7 application');
```

Again, query all records to check it:

```
select * from customers emit changes;
```

What happens this time?

You should see a table similar to the one shown below:

user_id	first_name	last_name	email	gender	club_status	comments
-1	Joey	Nelson	nelsonj@kuka.com	Male	silver	Rock band pop start
0	Melissa	Armstrong	melarm@yoga.io	Female	bronze	Yoga teacher
1	Rica	Blaisdell	rblaisdell10@rambler.ru	Female	bronze	Universal optimal hierarchy
2	Ruthie	Brockherst	rbrockherst1@ow.ly	Female	platinum	Reverse-engineered tangible interface
3	Mariejeanne	Cocci	mcocci2@techcrunch.com	Female	bronze	Multi-tiered bandwidth-monitored capability
4	Hashim	Rumke	hrumke3@sohu.com	Male	platinum	Self-enabling 24/7 firmware
5	Hansiain	Coda	hcoda4@senate.gov	Male	platinum	Centralised full-range approach
6	Robinet	Leheude	rleheude5@reddit.com	Female	platinum	Virtual upward-trending definition
7	Fay	Huc	fhuc6@quantcast.com	Female	bronze	Operative composite capacity
8	Patti	Rosten	prosten7@ihg.com	Female	silver	Integrated bandwidth-monitored instruction set
9	Even	Tinham	etinham8@facebook.com	Male	silver	Virtual full-range info-mediaries
10	Brena	Tollerton	btollerton9@furl.net	Female	silver	Diverse tangible methodology
11	Alexandro	Peeke-Vout	apeekevouta@freewebs.com	Male	gold	Ameliorated value-added orchestration
12	Sheryl	Hackwell	shackwellb@paginiegialle.it	Female	gold	Self-enabling global parallelism
13	Laney	Toopin	ltoopinc@icio.us	Female	platinum	Phased coherent alliance
14	Isabelita	Talboy	italboyd@imageshack.us	Female	gold	Cloned transitional synergy
15	Rodrique	Silverton	rsilvertone@umn.edu	Male	gold	Re-engineered static application
16	Clair	Vardy	cvardyf@reverbNation.com	Male	bronze	Expanded bottom-line Graphical User Interface
17	Brianna	Paradise	bparadiseg@nifty.com	Female	bronze	Open-source global toolset
18	Waldon	Keddey	wkeddeyh@weather.com	Male	gold	Business-focused multi-state functionalities
19	Josiah	Brockett	jbrocketti@com.com	Male	gold	Realigned didactic info-mediaries
20	Anselma	Rook	arookj@europa.eu	Female	gold	Cross-group 24/7 application

4.3 Identify the unhappy customers

Now that we have both our ratings and our continuously-updating table of customer data, we can join them together to find out details about the customers who are posting negative reviews, and see if any of them are our valued elite customers.

Back in KSQL, we start by finding just the low-scoring ratings:

```
select * from ratings
where stars < 3 and channel like 'iOS%'
emit changes limit 5;
```

⇒ Play around with the where clause conditions to experiment.

Now convert this test query into a persistent one (a persistent query is one which starts with create and continuously writes its' output into a topic in Kafka):

```
create stream poor_ratings as
select * from ratings
where stars < 3 and channel like 'iOS%';
```

Which of these low-score ratings was posted by an elite customer? To answer this we need to join the `poor_ratings` (derived) stream with the `customers` table:

```
create stream vip_poor_ratings as
  select r.user_id,
         c.first_name,
         c.last_name,
         c.club_status,
         r.stars
  from poor_ratings r
  left join customers c on r.user_id = c.user_id
  where lcase(c.club_status) = 'platinum';
```

Query that stream to see what's in it:

```
select * from vip_poor_ratings emit changes limit 10;
```

What do you think would happen if you went and changed the `club_status` of a customer while this join query is running? Let's try that!

4.4 Using tables as a Materialised Cache

One really interesting way to use ksqlDB is to think of a continuously-updating table as a kind of Materialised View or Cache (it isn't strictly either, technically speaking, but they are a good analogy here!) For example, we could also use one of our streams of ratings events to populate a table of aggregated, per-user, statistics:

```
create table rating_stats as
  select user_id,
         avg(stars) as avg_rating,
         collect_list(stars) as ratings,
         count(*) as num_ratings,
         max(rowtime) as last_rating_time
  from ratings
  group by user_id;
```

We can actually query this table in two different ways: 1. With a never-ending query, using `emit changes`, which will continuously output any changes in the table data back to our client.

As more of a "point look-up", which will simply give us the *current value* for a row in the table and then terminate. This should be familiar as it's how most databases work 😊

We sometimes refer to these 2 different ways to query as 'push' (the one which keeps sending change data back to us) and 'pull' (the moment-in-time lookup).

We can try it both ways with the table we just built, like this:

```
select * from rating_stats where user_id = 1 emit changes;
select * from rating_stats where user_id = 1;
```

See the difference?

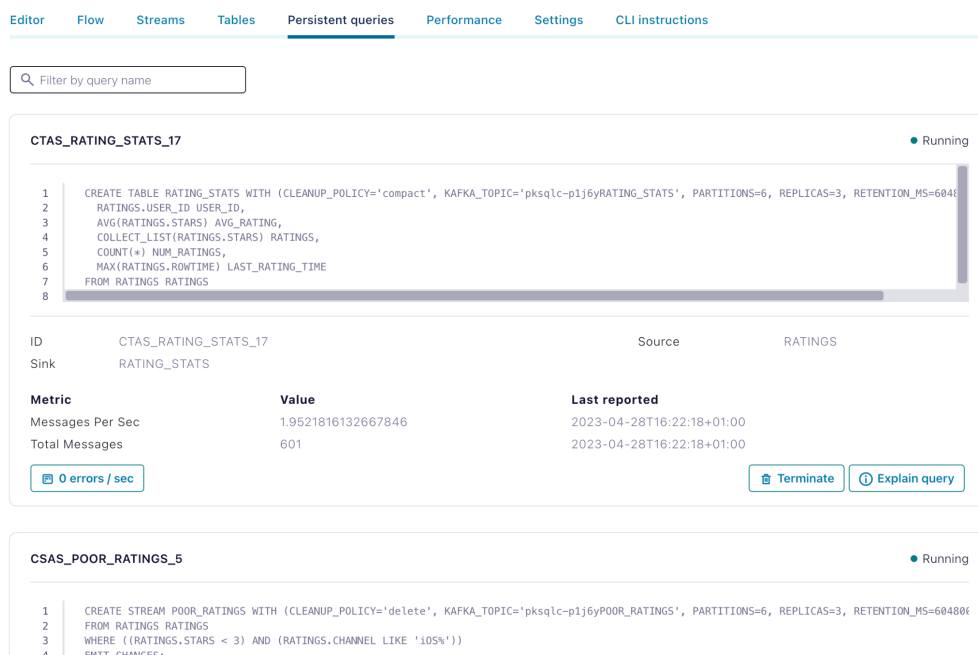
The first one (with **emit changes**) can be helpful when testing, or as the input to another **create stream as...** where you want the processing to continue for as long as new data is arriving. The second form (the lookup or 'pull query') can be useful if you have another application which wants to know the current value of something, perhaps in order to display it in a UI or use it in some other calculation. Because you can issue this query over ksqlDB's REST API it allows you to think of your ksqlDB apps as though they were like special microservices or cache servers, always running in the background, ready to serve up the latest state at any time.

4.5 Monitoring our Queries

So what's actually happening under the covers here ? Let's see all our running queries:

```
show queries;
explain <query_id>; (case sensitive!)
```

You can also see the persistent queries on the tab **Persistent queries**:



Editor Flow Streams Tables **Persistent queries** Performance Settings CLI instructions

Filter by query name

CTAS_RATING_STATS_17 Running

```
1 CREATE TABLE RATING_STATS WITH (CLEANUP_POLICY='compact', KAFKA_TOPIC='pksqlc-p1j6yRATING_STATS', PARTITIONS=6, REPLICAS=3, RETENTION_MS=604800)
2 RATING_STATS (USER_ID, USER_ID,
3 AVG(RATINGS.STARS) AVG_RATING,
4 COLLECT_LIST(RATINGS.STARS) RATINGS,
5 COUNT(*) NUM_RATINGS,
6 MAX(RATINGS.ROWTIME) LAST_RATING_TIME
7 FROM RATINGS RATINGS
8
```

ID	Sink	Source	Last reported
CTAS_RATING_STATS_17	RATING_STATS	RATINGS	

Metric	Value	Last reported
Messages Per Sec	1.9521816132667846	2023-04-28T16:22:18+01:00
Total Messages	601	2023-04-28T16:22:18+01:00

0 errors / sec Terminate Explain query

CSAS_POOR_RATINGS_5 Running

```
1 CREATE STREAM POOR_RATINGS WITH (CLEANUP_POLICY='delete', KAFKA_TOPIC='pksqlc-p1j6yPOOR_RATINGS', PARTITIONS=6, REPLICAS=3, RETENTION_MS=604800)
2 FROM RATINGS RATINGS
3 WHERE ((RATINGS.STARS < 3) AND (RATINGS.CHANNEL LIKE '105%'))
4 EMIT CHANGES;
```

4.5.1 View Consumer Lag for a Query

From the left-hand navigation menu select **Clients**, select **Consumers** tab. Now, try to find the one for our join query and click on it (all the names are prefixed with `confluent_ksql` plus the ID of the query, as shown in the output of explain queries).

<ul style="list-style-type: none"> Cluster Overview Dashboard Networking API Keys Cluster Settings Stream Lineage Stream Designer Topics ksqlDB Connectors Clients Schema Registry 	Clients		
	<div>Producers Consumers Consumer lag</div> <div>Search consumers</div>		
	Client ID ⓘ	Type	Consumption (last min)
	confluent-ksql-pksqlc-p1j6yquery_CSAS_POOR_RATIN...	ksqlDB	123B/s
	confluent-ksql-pksqlc-p1j6yquery_CSAS_VIP_POOR_R...	ksqlDB	43B/s
	confluent-ksql-pksqlc-p1j6yquery_CSAS_VIP_POOR_R...	ksqlDB	0B/s
	confluent-ksql-pksqlc-p1j6yquery_CTAS_RATING_STA...	ksqlDB	158B/s
			Number of topics

What do you see?

It's also possible (although not set up in this lab environment) to monitor a series of metrics for each running query.

5. Extra credit

Time permitting, let's explore the following ideas:

Which customers are so upset that they post multiple bad ratings in quick succession? Perhaps we want to route those complaints directly to our Customer Care team to do some outreach...

Unhappy VIP customers	Option: <u>Very</u> unhappy VIP customers
<pre>select first_name, last_name, count(*) as rating_count from vip_poor_ratings window tumbling (size 5 minutes) group by first_name, last_name having count(*) > 1 emit changes;</pre>	<pre>select first_name, last_name, count(*) as rating_count from vip_poor_ratings window tumbling (size 5 minutes) group by first_name, last_name having count(*) > 5 emit changes;</pre>

This may take a minute or two to return any data as we are now waiting for the random data generator which populates the original `ratings` to produce the needed set of output. And of course we could prefix this query with `create table unhappy_vips as ...` to continuously record the output.

6. Follow-on discussion points

6.1 Session properties

Investigate session properties with `show properties;` Although we won't be adjusting these today, the session properties mechanism is how you can temporarily adjust various performance settings for any subsequent queries you issue.

- [UDFs](#) (not yet available on Confluent Cloud)
- Testing tools
- [Mask](#) the actual user names in the output
- Explore and describe the available functions (not yet available on Confluent Cloud)
- Create a new stream over a topic that doesn't exist yet
- Use `insert...values` to write a couple of test records into this new topic join it to one of our existing streams or tables

7. Further resources

Don't forget to check out the #ksql channel on our [Community Slack group](https://slackpass.io/confluentcommunity) (<https://slackpass.io/confluentcommunity>).

Thank you and great job if you made this far! 🎉🌟👏💖

8. Deleting the resources created

Once you have completed your lab, it is recommended that you tear down all the resources created to avoid incurring additional charges on your Confluent Cloud account.

Note that if you do not delete the resources, they will be automatically deleted after 30 days or when your credits are consumed. If you wish to continue using Confluent Cloud after your trial period is over, you will need to set up a payment method. To do this, go to the top-right hand side menu, click **Billing & payment**, and select **Payment details & contacts**. Alternatively, you can contact us at cloud-support@confluent.io for further assistance.

8.1 Deleting the ksqlDB cluster

1. From the left-hand navigation in your Kafka cluster, navigate to **ksqlDB**
2. Select **Delete** under the Actions column
3. Type the cluster name and click **Continue**
4. All streams, tables and persistent queries created will be deleted

8.2 Deleting the Connect cluster

1. From the left-hand navigation in your Kafka cluster, navigate to **Connectors**
2. Select the connector created
3. Select tab **Settings**
4. Click **Delete connector**
5. Type the connector name and click **Confirm**

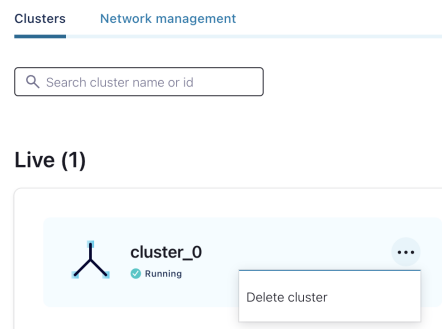
8.3 Deleting the Kafka cluster

1. From the top menu, click on the environment created to go back to the environment level



2. You will see the list of Kafka clusters created. On the one created for this workshop, click on the ... button and select **Delete cluster**

workshop-ksqldb



3. Type the cluster name and click **Continue**
4. All API keys, topics and data associated with them will also be deleted

8.4 Deleting the Environment

1. From the left-hand navigation menu select **Environments**
2. Select the environment created for this workshop **workshop-ksqldb**
3. On the bottom-left of the page click **Delete Environment**
4. Type the environment name and click **Continue**
5. The Schema Registry cluster associated with it will also be deleted