Quick Start for Confluent Platform (Local install)

Use this quick start to get up and running with Confluent Platform and its main components in a development environment. This quick start uses Confluent Control Center included in Confluent Platform for topic management and event stream processing using ksqlDB.

In this quick start, you create Apache Kafka® topics, use Kafka Connect to generate mock data to those topics, and create ksqlDB streaming queries on those topics. You then go to Control Center to monitor and analyze the streaming queries.

Prerequisites:

- Internet connectivity.
- Operating System currently supported by Confluent Platform.
- A supported version of Java downloaded and installed.

  Java 8 and Java 11 are supported in this version of Confluent Platform (Java 9 and 10 are not supported).

Step 1: Download and Start Confluent Platform

1. Go to the downloads page.
2. Scroll to the **Download Confluent Platform** section and provide the following:
   - Email: Your email address
   - Format: `deb`, `rpm`, `tar`, or `zip`
3. Click **DOWNLOAD FREE**.
4. Decompress the file. You should have the directories, such as `bin` and `etc`.
5. Set the environment variable for the Confluent Platform directory.

```
export CONFLUENT_HOME=<path-to-confluent>
```

6. Add the Confluent Platform `bin` directory to your PATH.

```
export PATH=$PATH:$CONFLUENT_HOME/bin
```

7. Install the Kafka Connect Datagen source connector using the Confluent Hub client. This connector generates mock data for demonstration purposes and is not suitable for production. Confluent Hub is an online library of pre-packaged and ready-to-install extensions or add-ons for Confluent Platform and Kafka.

```
confluent-hub install \
   --no-prompt confluentinc/kafka-connect-datagen:latest
```

8. Start Confluent Platform using the Confluent CLI confluent local services start command. This command starts all of the Confluent Platform components, including Kafka, ZooKeeper, Schema Registry, HTTP REST Proxy for Kafka, Kafka Connect, ksqlDB, and Control Center.

Important

The confluent local commands are intended for a single-node development environment and are not suitable for a production environment. The data that are produced are transient and are intended to be temporary.

```
confluent local services start
```

Your output should resemble:

```
Starting Zookeeper
Zookeeper is [UP]
Starting Kafka
Kafka is [UP]
Starting Schema Registry
Schema Registry is [UP]
Starting Kafka REST
Kafka REST is [UP]
Starting Connect
Connect is [UP]
Starting KSQL Server
KSQL Server is [UP]
Starting Control Center
Control Center is [UP]
```

Step 2: Create Kafka Topics

In this step, you create Kafka topics using 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.

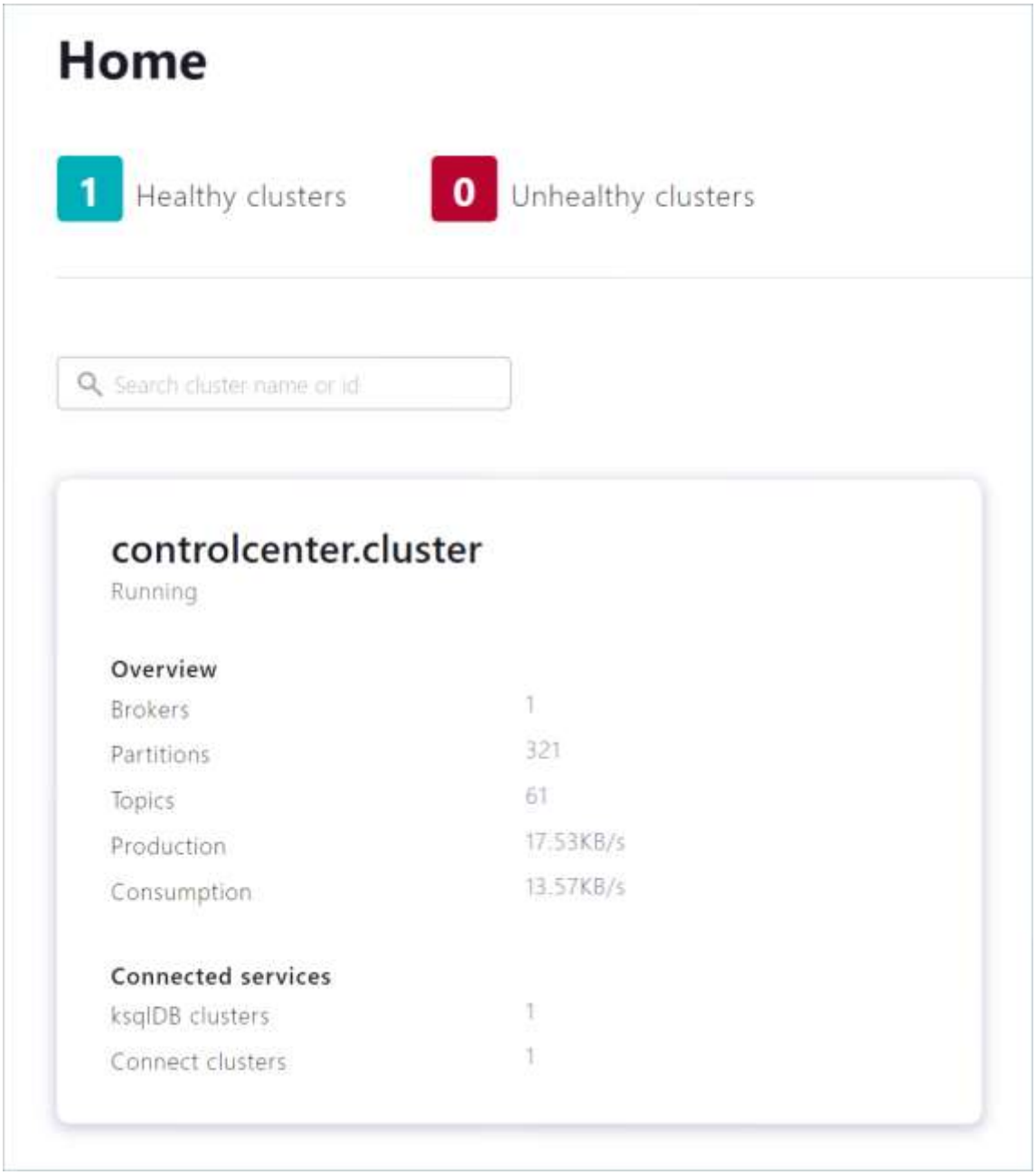
   If you installed Confluent Platform on a different host, replace `localhost` with the host name in the address.

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

Note

Control Center won't connect to ksqlDB if Control Center isn't open and running in a `localhost` browser session.

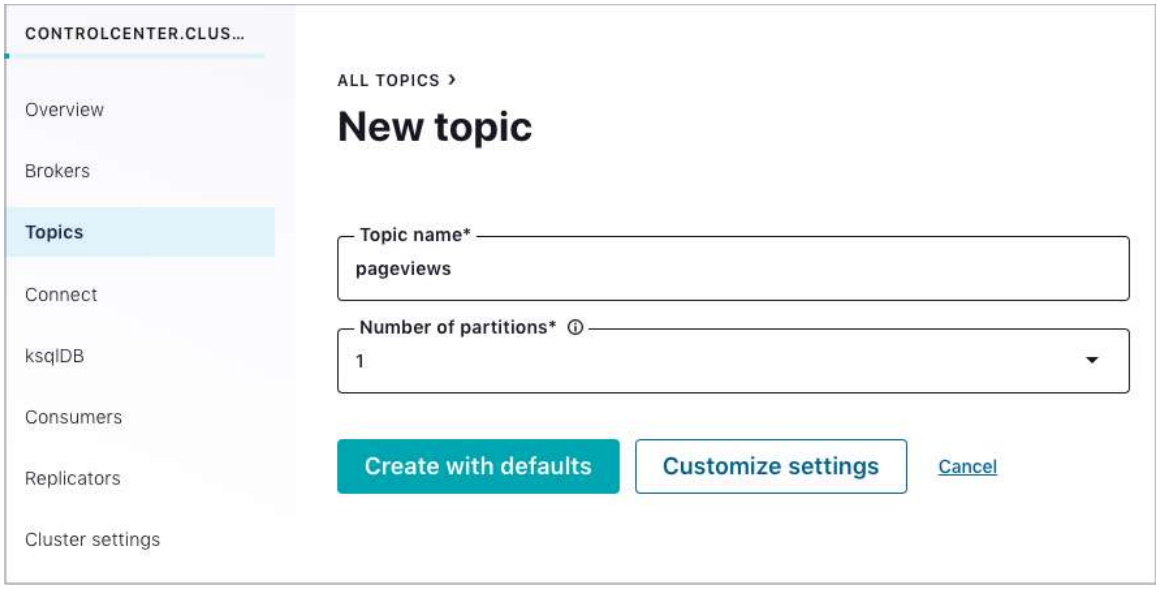
2. Click the **controlcenter.cluster** tile.



3. In the navigation bar, click **Topics** to open the topics list, and then click **Add a topic**.

4. In the Topic name field, specify `pageviews` and click **Create with defaults**.

   Note that topic names are case-sensitive.



5. In the navigation bar, click **Topics** to open the topics list, and then click **Add a topic**.
6. In the Topic name field, specify `users` and click **Create with defaults**.

Step 3: 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`.

Tip

The Kafka Connect Datagen connector was installed manually in Step 1: Download and Start Confluent Platform.

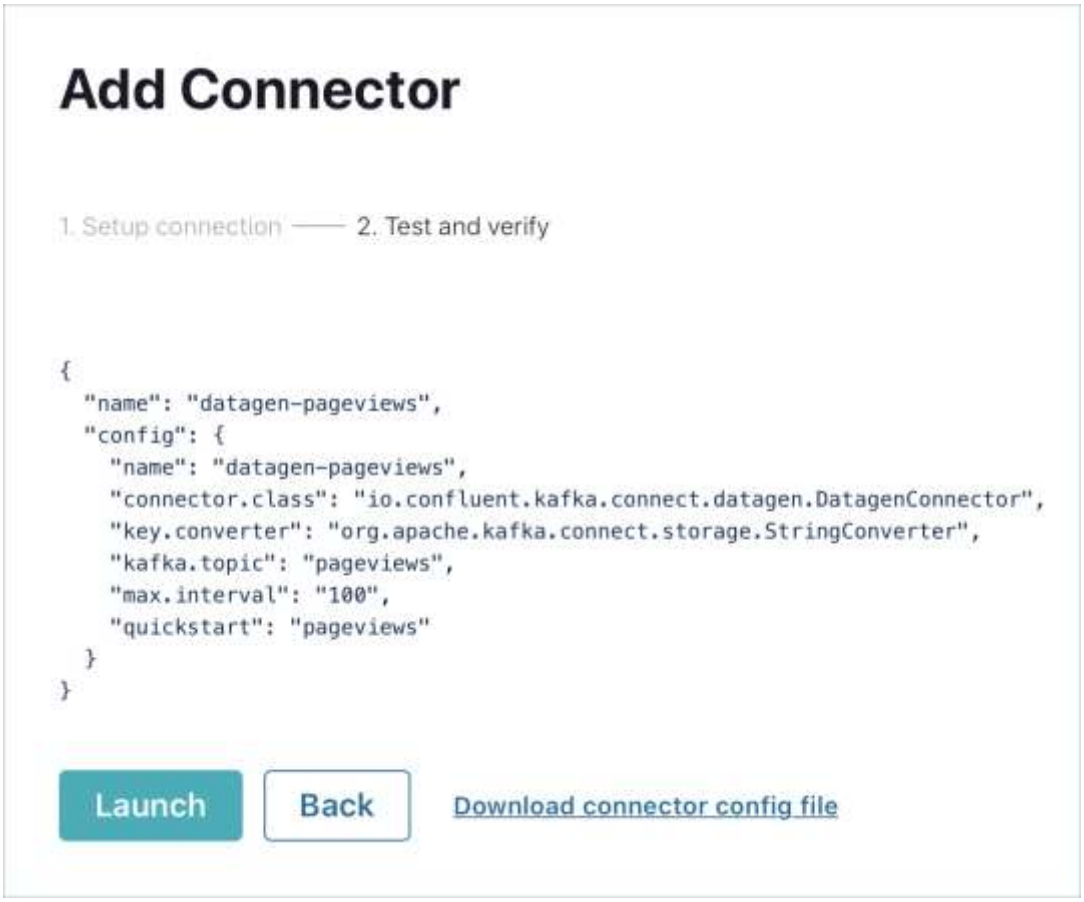
1. Run the first instance of the Kafka Connect Datagen connector to produce Kafka data to the `pageviews` topic in AVRO format.

    1. In the navigation bar, click **Connect**.
    2. Click the `connect-default` cluster in the **Connect Clusters** list.
    3. Click **Add connector**.
    4. Select the `DatagenConnector` tile.

    5. In the **Name** field, enter `datagen-pageviews` as the name of the connector.
    6. Enter the following configuration values:

        - **Key converter class:** `org.apache.kafka.connect.storage.StringConverter`.
        - **kafka.topic:** `pageviews`.
        - **max.interval:** `100`.
        - **quickstart:** `pageviews`.

7. Click **Next**.
8. Review the connector configuration and click **Launch**.



2. Run the second instance of the Kafka Connect Datagen connector to produce Kafka data to the `users` topic in AVRO format.

    1. Click **Add connector**.
    2. Select the `DatagenConnector` tile.
    3. In the **Name** field, enter `datagen-users` as the name of the connector.
    4. Enter the following configuration values:

        - **Key converter class:** `org.apache.kafka.connect.storage.StringConverter`
        - **kafka.topic:** `users`
        - **max.interval:** `1000`
        - **quickstart:** `users`
    5. Click **Next**.
    6. Review the connector configuration and click **Launch**.

Step 4: Create and Write to a Stream and Table using ksqlDB

Tip

You can also run these commands using the ksqlDB CLI from your terminal with this command: `<path-to-confluent>/bin/ksql http://localhost:8088`.

Create Streams and Tables

In this step, you use ksqlDB to create a stream for the `pageviews` topic and a table for the `users` topic.

1. In the navigation bar, click **ksqlDB**.
2. Select the `ksqlDB` application.
3. Copy the following code into the editor window and click **Run query** to create the `pageviews` stream. Stream names are not case-sensitive.

```
CREATE STREAM pageviews WITH (KAFKA_TOPIC='pageviews', VALUE_FORMAT='AVRO');
```

4. Copy the following code into the editor window and click **Run query** to create the `users` table. Table names are not case-sensitive.

```
CREATE TABLE users (id VARCHAR PRIMARY KEY)
  WITH (KAFKA_TOPIC='users', VALUE_FORMAT='AVRO');
```

Write Queries

In this step, you create ksqlDB queries against the stream and the table you created above.

1. In the **Editor** tab, click **Add query properties** to add a custom query property.
2. Set the `auto.offset.reset` parameter to `Earliest`.

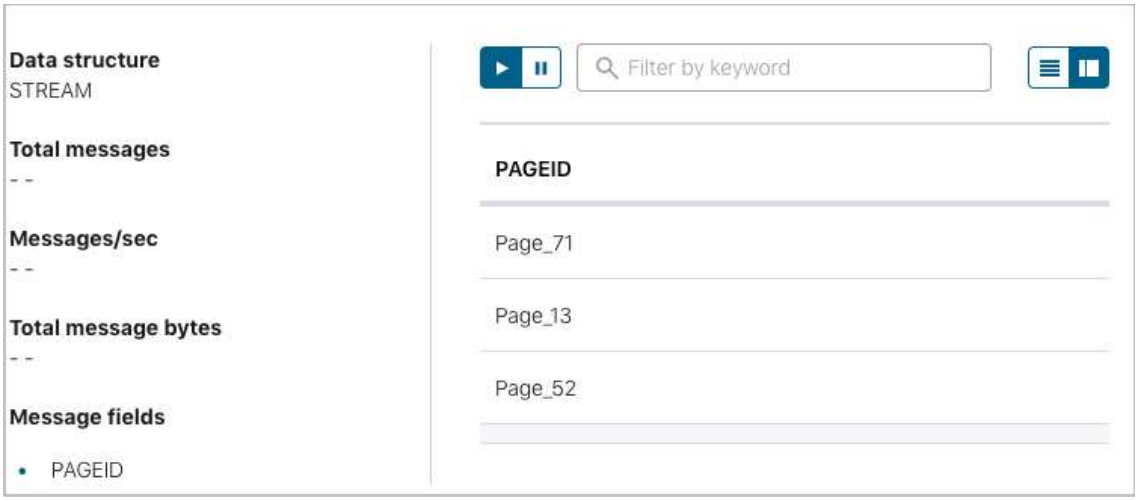
   The setting instructs ksqlDB queries to read all available topic data from the beginning. This configuration is used for each subsequent query.
3. Create the following queries.

   1. Click **Stop** to stop the current running query.
   2. Create a non-persistent query that returns data from a stream with the results limited to a maximum of three rows:

      Enter the following query in the editor:

      ```
      SELECT pageid FROM pageviews EMIT CHANGES LIMIT 3;
      ```

   3. Click **Run query**. Your output should resemble:



Click the **Card view** or **Table view** icon to change the output layout.

4. Create a persistent query (as a stream) that filters the `PAGEVIEWS` stream for female users. The results from this query are written to the Kafka `PAGEVIEWS_FEMALE` topic:

   Enter the following query in the editor:

```
CREATE STREAM pageviews_female
  AS SELECT users.id AS userid, pageid, regionid
  FROM pageviews LEFT JOIN users ON pageviews.userid = users.id
  WHERE gender = 'FEMALE'
  EMIT CHANGES;
```

5. Click **Run query**. Your output should resemble:


```
{
  "@type": "currentStatus",
  "statementText": "CREATE STREAM PAGEVIEWS_FEMALE WITH (KAFKA_TOPIC='PAGEVIEWS_
  "commandId": "stream/`PAGEVIEWS_FEMALE`/create",
  "commandStatus": {
    "status": "SUCCESS",
    "message": "Created query with ID CSAS_PAGEVIEWS_FEMALE_0"
  },
  "commandSequenceNumber": 6,
  "warnings": []
}
```


6. Create a persistent query where `REGIONID` ends with `8` or `9`. Results from this query are written to the Kafka topic named `pageviews_enriched_r8_r9` as explicitly specified in the query:

   Enter the following query in the editor:

```
CREATE STREAM pageviews_female_like_89
  WITH (KAFKA_TOPIC='pageviews_enriched_r8_r9', VALUE_FORMAT='AVRO')
```

```
  AS SELECT * FROM pageviews_female
  WHERE regionid LIKE '%_8' OR regionid LIKE '%_9'
  EMIT CHANGES;
```

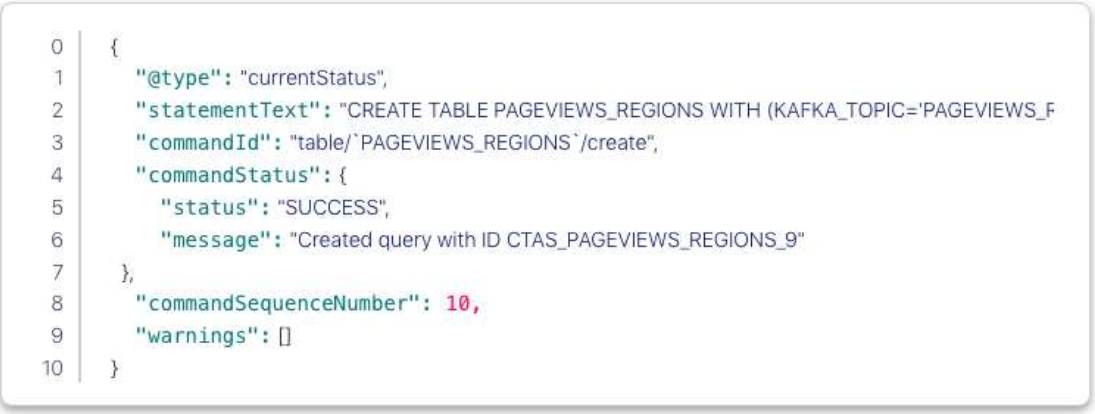
7. Click **Run query**. Your output should resemble:

8. 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`:

   Enter the following query in the editor:

```
CREATE TABLE pageviews_regions WITH (KEY_FORMAT='JSON')
  AS SELECT gender, regionid, COUNT(*) AS numusers
  FROM pageviews LEFT JOIN users ON pageviews.userid = users.id
  WINDOW TUMBLING (SIZE 30 SECOND)
  GROUP BY gender, regionid
  HAVING COUNT(*) > 1
  EMIT CHANGES;
```

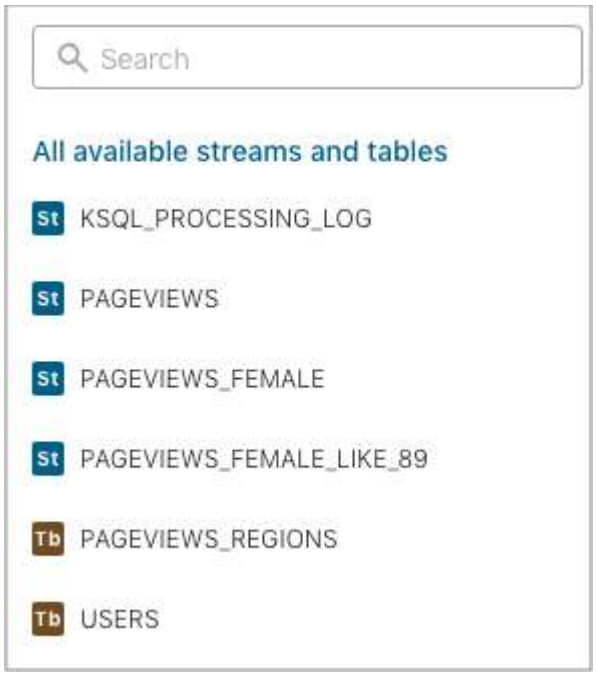
9. Click **Run query**. Your output should resemble:



10. Click the **Persistent queries** tab. You should see the following persisted queries:

    - PAGEVIEWS_FEMALE
    - PAGEVIEWS_FEMALE_LIKE_89
    - PAGEVIEWS_REGIONS

11. Click the **Editor** tab. The **All available streams and tables** pane shows all of the streams and tables that you can access.



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

Run Queries

In this step, you run the ksqlDB queries you save as streams and tables above in the previous section.

1. In the **Streams** tab, select the `PAGEVIEWS_FEMALE` stream.
2. Click **Query stream**.

   The editor opens, and streaming output of the query displays.
3. Click **Stop** to stop the output generation.
4. In the **Tables** tab, select `PAGEVIEWS_REGIONS` table.
5. Click **Query table**.

   The editor opens, and streaming output of the query displays.
6. Click **Stop** to stop the output generation.

Step 5: Monitor Consumer Lag

1. In the navigation bar, click **Consumers** to view the consumers created by ksqlDB.
2. Click the consumer group ID to view details for the `_confluent-ksql-default_query_CSAS_PAGEVIEWS_FEMALE_5` consumer group.

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

Step 6: Stop Confluent Platform

When you are done working with the local install, you can stop Confluent Platform.

1. Stop Confluent Platform using the Confluent CLI confluent local services connect stop command.

```
confluent local services stop
```

2. Destroy the data in the Confluent Platform instance with the confluent local destroy command.

```
confluent local destroy
```

Troubleshooting

If you encountered any issues while going through the quickstart workflow, review the following resolutions before trying the steps again.

Issue: Cannot locate the Datagen connector

Resolution: Verify that you have added the location of the Confluent Platform `bin` directory to your `PATH` as described in Step 1: Download and Start Confluent Platform:

```
export PATH=<path-to-confluent>/bin:$PATH
```

Resolution: Verify the DataGen connector is installed and running.

Ensure that the `kafka-connect-datagen` is installed and running as described in Step 1: Download and Start Confluent Platform.

```
confluent-hub install --no-prompt confluentinc/kafka-connect-datagen:latest
```

Your output should resemble:

```
Running in a "--no-prompt" mode
...
Completed
```

Resolution: Check the connect logs for `Datagen` using the Confluent CLI confluent local services connect log command.

```
confluent local services connect log | grep -i Datagen
```

Your output should resemble:

```
[2019-04-18 14:21:08,840] INFO Loading plugin from: /Users/user.name/Confluent/con
fluent-version/share/confluent-hub-components/confluentinc-kafka-connect-datagen (
org.apache.kafka.connect.runtime.isolation.DelegatingClassLoader:215)
[2019-04-18 14:21:08,894] INFO Registered loader: PluginClassLoader{pluginLocation
=file:/Users/user.name/Confluent/confluent-version/share/confluent-hub-components/
confluentinc-kafka-connect-datagen/} (org.apache.kafka.connect.runtime.isolation.D
elegatingClassLoader:238)
[2019-04-18 14:21:08,894] INFO Added plugin 'io.confluent.kafka.connect.datagen.Da
tagenConnector' (org.apache.kafka.connect.runtime.isolation.DelegatingClassLoader:
167)
[2019-04-18 14:21:09,882] INFO Added aliases 'DatagenConnector' and 'Datagen' to p
lugin 'io.confluent.kafka.connect.datagen.DatagenConnector' (org.apache.kafka.conn
ect.runtime.isolation.DelegatingClassLoader:386)
```

Resolution: Verify the `.jar` file for `kafka-connect-datagen` has been added and is present in the `lib` subfolder.

```
ls $CONFLUENT_HOME/share/confluent-hub-components/confluentinc-kafka-connect-datag
en/lib/
```

Your output should resemble:

```
...
kafka-connect-datagen-0.1.0.jar
...
```

Resolution: Verify the plugin exists in the connector path.

When you installed the `kafka-connect-datagen` file from Confluent hub, the installation directory is added to the plugin path of several properties files:

```
 Adding installation directory to plugin path in the following files:
/Users/user.name/Confluent/confluent-version/etc/kafka/connect-distributed.properties
/Users/user.name/Confluent/confluent-version/etc/kafka/connect-standalone.properties
/Users/user.name/Confluent/confluent-version/etc/schema-registry/connect-avro-distributed.properties
/Users/user.name/Confluent/confluent-version/etc/schema-registry/connect-avro-standalone.properties
...
```

You can use any of them to check the connector path. This example uses the `connect-avro-distributed.properties` file.

```
grep plugin.path $CONFLUENT_HOME/etc/schema-registry/connect-avro-distributed.properties
```

Your output should resemble:

```
plugin.path=share/java,/Users/user.name/Confluent/confluent-version/share/confluent-hub-components
```

Confirm its contents are present:

```
ls $CONFLUENT_HOME/share/confluent-hub-components/confluentinc-kafka-connect-datagen
```

Your output should resemble:

```
assets   doc  lib  manifest.json
```

Issue: Stream-Stream joins error

An error states Stream-Stream joins must have a `WITHIN` clause specified. This error can occur if you created both `pageviews` and `users` as streams by mistake.

Resolution: Ensure that you created a *stream* for `pageviews`, and a *table* for `users` in Step 4: Create and Write to a Stream and Table using ksqlDB.

Issue: Unable to successfully complete ksqlDB query steps

Java errors or other severe errors were encountered.

Resolution: Ensure you are on an Operating System currently supported by Confluent Platform.