

# Quick Start for Confluent

## Platform - Community Components (Local install)

Use this quick start to get up and running with Confluent Platform and Confluent Community components in a development environment.

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.

This quick start leverages the Confluent Platform CLI, the Apache Kafka® CLI, and the ksqlDB CLI.

### Prerequisites:

- Internet connectivity.
- [Operating System](#) currently supported by Confluent Platform.
- [A supported version of Java](#) downloaded and installed.

## Step 1: Download and Start Confluent Platform

1. Go to the [downloads page](#).
2. Scroll to the **Download Confluent Platform** section and click the **here** link to download the free community features.
3. Provide the following:
  - Email: Your email address
  - File Type: , , , or
  - Agree to the terms of the Confluent Community License Agreement.
4. Click **DOWNLOAD**.
5. Decompress the file. You should have these directories:

Folder	Description
/bin/	Driver scripts for starting and stopping services
/etc/	Configuration files
/lib/	Systemd services
/logs/	Log files
/share/	Jars and licenses
/src/	Source files that require a platform-dependent build

- Set the following shell variables:

```
export CONFLUENT_HOME=<path-to-confluent>
export PATH="${CONFLUENT_HOME}/bin:$PATH"
```

- Install the Confluent Hub client. This is used in the next step to install the free and open source `kafka-source-datagen` connector.
- Install the Confluent CLI, `confluent`, using the following script.

On Microsoft Windows, an appropriate Linux environment may need to be installed in order to have the `curl` and `sh` commands available, such as the [Windows Subsystem for Linux](#).

```
curl -L --http1.1 https://cnfl.io/cli | sh -s -- -b $CONFLUENT_HOME/bin
```

- 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
```

- Start Confluent Platform using the `confluent local services start` command. This command will start all of the Confluent Platform components, including Kafka, ZooKeeper, Schema Registry, HTTP REST Proxy for Kafka, Kafka Connect, and KSQLDB.

```
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]
```

## Step 2: Create Kafka Topics

In this step, you create Kafka topics using the Kafka CLI.

1. Create a topic named `users`:

```
kafka-topics --create \  
  --bootstrap-server localhost:9092 \  
  --replication-factor 1 \  
  --partitions 1 \  
  --topic users
```

2. Create a topic named `pageviews`:

```
kafka-topics --create \  
  --bootstrap-server localhost:9092 \  
  --replication-factor 1 \  
  --partitions 1 \  
  --topic pageviews
```

## 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`.

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

```
curl -L -O -H 'Accept: application/vnd.github.v3.raw' \  
  https://api.github.com/repos/confluentinc/kafka-connect-datagen/contents/con  
fig/connector_pageviews_cos.config  
  
curl -X POST -H "Content-Type: application/json" \  
  --data @connector_pageviews_cos.config http://localhost:8083/connectors
```

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

```
curl -L -O -H 'Accept: application/vnd.github.v3.raw' \  
  https://api.github.com/repos/confluentinc/kafka-connect-datagen/contents/con  
fig/connector_users_cos.config  
  
curl -X POST -H "Content-Type: application/json" \  
  --data @connector_users_cos.config http://localhost:8083/connectors
```

```
--data @connector_users_cos.config http://localhost:8083/connectors
```

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

In this step, you create streams, tables, and queries using ksqlDB SQL.

### Create Streams and Tables

1. Start the ksqlDB CLI in your terminal with this command.

```
LOG_DIR=$CONFLUENT_HOME/ksql_logs ksql
```

#### Important

By default ksqlDB attempts to store its logs in a directory called `logs` that is relative to the location of the `ksql` executable. For example, if `ksql` is installed at `/usr/local/bin/ksql`, then it would attempt to store its logs in `/usr/local/logs`. If you are running `ksql` from the default Confluent Platform location, `$CONFLUENT_HOME/bin`, you must override this default behavior by using the `LOG_DIR` variable.

2. Create a stream `PAGEVIEWS` from the Kafka topic `pageviews`, specifying the `value_format` of `AVRO`:

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

3. Create a table `USERS` with several columns from the Kafka topic `users`, with the `value_format` of `AVRO`:

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

### Write Queries

In this step, you run ksqlDB SQL queries.

1. Set the `auto.offset.reset` query property to ``earliest`.`

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

```
SET 'auto.offset.reset'='earliest';
```

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

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

Your output should resemble:

```
Page_45  
Page_38  
Page_11  
LIMIT reached  
Query terminated
```

3. 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:

```
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;
```

4. 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:

```
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;
```

5. 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`:

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

```
EMIT CHANGES;
```

## Examine Streams, Tables, and Queries

- List the streams:

```
SHOW STREAMS;
```

- List the tables:

```
SHOW TABLES;
```

- View the details of a stream or a table:

```
DESCRIBE <stream-or-table-name> EXTENDED;
```

For example, to view the details of the `users` table:

```
DESCRIBE USERS EXTENDED;
```

- List the running queries:

```
SHOW QUERIES;
```

- Review the query execution plan:

Get a Query ID from the output of `SHOW QUERIES` and run `EXPLAIN` to view the query execution plan for the Query ID:

```
EXPLAIN <Query ID>;
```

## Step 5: Monitor Streaming Data

Now you can monitor the running queries created as streams or tables.

- The following query returns the page view information of female users:

```
SELECT * FROM pageviews_female EMIT CHANGES LIMIT 5;
```

- The following query returns the page view information of female users in the regions whose `regionid` ends with `8` or `9`:

```
SELECT * FROM pageviews_female_like_89 EMIT CHANGES LIMIT 5;
```

- The following query returns the page view counts for each region and gender combination in a tumbling window of 30 seconds. To see table updates, let the query run for a few seconds. Press Ctrl+C to stop the query.

```
SELECT * FROM pageviews_regions EMIT CHANGES;
```

## 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
```

You can start the local install of Confluent Platform again with the [confluent local services start](#) command.

## 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 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/confluent-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.DelegatingClassLoader:238)
[2019-04-18 14:21:08,894] INFO Added plugin 'io.confluent.kafka.connect.datagen.DatagenConnector' (org.apache.kafka.connect.runtime.isolation.DelegatingClassLoader:167)
[2019-04-18 14:21:09,882] INFO Added aliases 'DatagenConnector' and 'Datagen' to plugin 'io.confluent.kafka.connect.datagen.DatagenConnector' (org.apache.kafka.connect.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-datagen/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.

ksqlDB errors were encountered.

**Resolution:** Review the help in the ksqlDB CLI for successful command tips and links to more documentation.

```
ksql> help
```