## **Try it**

#### **1-Initialize the project**

* To get started, make a new directory anywhere you’d like for this project:

mkdir transform-stream && cd transform-stream

* Then make the following directories to set up its structure:

mkdir src test

#### **2-Get Confluent Platform**

* Next, create the following docker-compose.yml file to obtain Confluent Platform:

---

version: '2'

services:

zookeeper:

image: confluentinc/cp-zookeeper:5.3.0

hostname: zookeeper

container\_name: zookeeper

ports:

- "2181:2181"

environment:

ZOOKEEPER\_CLIENT\_PORT: 2181

ZOOKEEPER\_TICK\_TIME: 2000

broker:

image: confluentinc/cp-enterprise-kafka:5.3.0

hostname: broker

container\_name: broker

depends\_on:

- zookeeper

ports:

- "29092:29092"

environment:

KAFKA\_BROKER\_ID: 1

KAFKA\_ZOOKEEPER\_CONNECT: 'zookeeper:2181'

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://broker:9092,PLAINTEXT\_HOST://localhost:29092

KAFKA\_METRIC\_REPORTERS: io.confluent.metrics.reporter.ConfluentMetricsReporter

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS: 0

CONFLUENT\_METRICS\_REPORTER\_BOOTSTRAP\_SERVERS: broker:9092

CONFLUENT\_METRICS\_REPORTER\_ZOOKEEPER\_CONNECT: zookeeper:2181

CONFLUENT\_METRICS\_REPORTER\_TOPIC\_REPLICAS: 1

CONFLUENT\_METRICS\_ENABLE: 'true'

CONFLUENT\_SUPPORT\_CUSTOMER\_ID: 'anonymous'

schema-registry:

image: confluentinc/cp-schema-registry:5.3.0

hostname: schema-registry

container\_name: schema-registry

depends\_on:

- zookeeper

- broker

ports:

- "8081:8081"

environment:

SCHEMA\_REGISTRY\_HOST\_NAME: schema-registry

SCHEMA\_REGISTRY\_KAFKASTORE\_CONNECTION\_URL: 'zookeeper:2181'

ksql-server:

image: confluentinc/cp-ksql-server:5.3.0

hostname: ksql-server

container\_name: ksql-server

depends\_on:

- broker

- schema-registry

ports:

- "8088:8088"

environment:

KSQL\_CONFIG\_DIR: "/etc/ksql"

KSQL\_LOG4J\_OPTS: "-Dlog4j.configuration=file:/etc/ksql/log4j-rolling.properties"

KSQL\_BOOTSTRAP\_SERVERS: "broker:9092"

KSQL\_HOST\_NAME: ksql-server

KSQL\_LISTENERS: "http://0.0.0.0:8088"

KSQL\_CACHE\_MAX\_BYTES\_BUFFERING: 0

KSQL\_KSQL\_SCHEMA\_REGISTRY\_URL: "http://schema-registry:8081"

KSQL\_PRODUCER\_INTERCEPTOR\_CLASSES: "io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor"

KSQL\_CONSUMER\_INTERCEPTOR\_CLASSES: "io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor"

ksql-cli:

image: confluentinc/cp-ksql-cli:5.3.0

container\_name: ksql-cli

depends\_on:

- broker

- ksql-server

entrypoint: /bin/sh

tty: true

volumes:

- ./src:/opt/app/src

- ./test:/opt/app/test

* And launch it by running:

docker-compose up

#### **3-Write the program interactively using the CLI**

* To begin developing interactively, open up the KSQL CLI:

docker exec -it ksql-cli ksql http://ksql-server:8088

* First, you’ll need to create a Kafka topic and stream to represent the publications. The following creates both in one shot:

CREATE STREAM raw\_movies (id int, title varchar, genre varchar)

WITH (kafka\_topic='movies', partitions=1, key='id', value\_format = 'avro');

* Then produce the following events to the stream:

INSERT INTO raw\_movies (id, title, genre) VALUES (294, 'Die Hard::1988', 'action');

INSERT INTO raw\_movies (id, title, genre) VALUES (354, 'Tree of Life::2011', 'drama');

INSERT INTO raw\_movies (id, title, genre) VALUES (782, 'A Walk in the Clouds::1995', 'romance');

INSERT INTO raw\_movies (id, title, genre) VALUES (128, 'The Big Lebowski::1998', 'comedy');

* Now that you have stream with some events in it, let’s read them out. The first thing to do is set the following properties to ensure that you’re reading from the beginning of the stream:

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

* Let’s break apart the title field and extract the year that the movie was published into its own column. Issue the following transient query. This will block and continue to return results until it’s limit is reached or you tell it to stop.

SELECT id, split(title, '::')[0] as title, split(title, '::')[1] AS year, genre FROM raw\_movies LIMIT 4;

* This should yield the following output:

294 | Die Hard | 1988 | action

354 | Tree of Life | 2011 | drama

782 | A Walk in the Clouds | 1995 | romance

128 | The Big Lebowski | 1998 | comedy

Limit Reached

Query terminated

* Since the output looks right, the next step is to make the query continuous. Issue the following to create a new stream that is continously populated by its query:

CREATE STREAM movies WITH (kafka\_topic = 'parsed\_movies', partitions = 1) AS

SELECT id,

split(title, '::')[0] as title,

CAST(split(title, '::')[1] AS INT) AS year,

genre

FROM raw\_movies;

* To check that it’s working, print out the contents of the output stream’s underlying topic:

PRINT 'parsed\_movies' FROM BEGINNING LIMIT 4;

This should yield the following output:

Format:AVRO

6/26/19 5:58:26 PM UTC, 294, {"ID": 294, "TITLE": "Die Hard", "YEAR": 1988, "GENRE": "action"}

6/26/19 5:58:27 PM UTC, 354, {"ID": 354, "TITLE": "Tree of Life", "YEAR": 2011, "GENRE": "drama"}

6/26/19 5:58:27 PM UTC, 782, {"ID": 782, "TITLE": "A Walk in the Clouds", "YEAR": 1995, "GENRE": "romance"}

6/26/19 5:58:27 PM UTC, 128, {"ID": 128, "TITLE": "The Big Lebowski", "YEAR": 1998, "GENRE": "comedy"}

#### **4-Write your statements to a file**

* Now that you have a series of statements that’s doing the right thing, the last step is to put them into a file so that they can be used outside the CLI session. Create a file at src/statements.sql with the following content:

CREATE STREAM raw\_movies (id int, title varchar, genre varchar)

WITH (kafka\_topic='movies', partitions=1, key='id', value\_format = 'avro');

CREATE STREAM movies WITH (kafka\_topic = 'parsed\_movies', partitions = 1) AS

SELECT id,

split(title, '::')[0] as title,

CAST(split(title, '::')[1] AS INT) AS year,

genre

FROM raw\_movies;

## **Test it**

#### **1-Create the test data**

Create a file at test/input.json with the inputs for testing:

{

"inputs": [

{

"topic": "movies",

"key": 294,

"value": {

"id": 294,

"title": "Die Hard::1988",

"genre": "action"

}

},

{

"topic": "movies",

"key": 354,

"value": {

"id": 354,

"title": "Tree of Life::2011",

"genre": "drama"

}

},

{

"topic": "movies",

"key": 782,

"value": {

"id": 782,

"title": "A Walk in the Clouds::1995",

"genre": "romance"

}

},

{

"topic": "movies",

"key": 128,

"value": {

"id": 128,

"title": "The Big Lebowski::1998",

"genre": "comedy"

}

}

]

}

* Similarly, create a file at test/output.json with the expected outputs:

{

"outputs": [

{

"topic": "parsed\_movies",

"key": 294,

"value": {

"ID": 294,

"TITLE": "Die Hard",

"YEAR": 1988,

"GENRE": "action"

}

},

{

"topic": "parsed\_movies",

"key": 354,

"value": {

"ID": 354,

"TITLE": "Tree of Life",

"YEAR": 2011,

"GENRE": "drama"

}

},

{

"topic": "parsed\_movies",

"key": 782,

"value": {

"ID": 782,

"TITLE": "A Walk in the Clouds",

"YEAR": 1995,

"GENRE": "romance"

}

},

{

"topic": "parsed\_movies",

"key": 128,

"value": {

"ID": 128,

"TITLE": "The Big Lebowski",

"YEAR": 1998,

"GENRE": "comedy"

}

}

]

}

#### **2-Invoke the tests**

* Lastly, invoke the tests using the test runner and the statements file that you created earlier:

docker exec ksql-cli ksql-test-runner -i /opt/app/test/input.json -s opt/app/src/statements.sql -o /opt/app/test/output.json

* Which should pass:

>>> Test passed!

## **Take it to production**

#### **1-Send the statements to the REST API**

* Launch your statements into production by sending them to the REST API with the following command:

statements=$(< src/statements.sql) && \

echo '{"ksql":"'$statements'", "streamsProperties": {}}' | \

curl -X "POST" "http://localhost:8088/ksql" \

-H "Content-Type: application/vnd.ksql.v1+json; charset=utf-8" \

-d @- | \

jq