

[Return to Classroom](#)[DISCUSS ON STUDENT HUB](#)

# Optimizing Public Transportation

## REVIEW

### CODE REVIEW 17

## HISTORY

### Requires Changes

#### 1 specification requires changes

Great job, you are almost there! <sup>100</sup> Clearly, you have acquired all the important concepts from this project. You only need to make some modifications and then you are ready to go.

Tip: If you want to learn more about the python kafka client, please check out [this tutorial](#). I would suggest that you could read [this article](#) about how to do Kafka Structured Streaming production monitoring.

### Kafka Producer

Using the Kafka Topics CLI, topics appear for arrivals on each train line in addition to the turnstiles for each of those stations.

Good job setting up the required topic names for each producer.

```
root@41735495f605:/home/workspace/home# kafka-topics --list --zookeeper localhost:2181
__confluent.support.metrics
__consumer_offsets
__confluent-ksql-default__command_topic
__confluent-metrics
__schemas
com.udacity.streams.clickevents
com.udacity.streams.pages
com.udacity.streams.purchases
com.udacity.streams.users
connect-configs
connect-offsets
connect-status
org.chicago.cta.station.arrivals.v1
org.chicago.cta.turnstile.v1
org.chicago.cta.weather.v1
postgres-stations
root@41735495f605:/home/workspace/home#
```

Using the Kafka Topics CLI, messages continuously appear for each station on the train line, for both arrivals and turnstile actions.

All turnstiles and arrivals messages are correctly produced. Well done!

```
root@41735495f605:/home/workspace/home# kafka-avro-console-consumer --topic org.chicago.cta.turnstile.v1 --from-beginning --bootstrap-server localhost:9092 --max-messages 10
{"station_id":40070,"station_name":"Jackson","line":"blue"}
{"station_id":40070,"station_name":"Jackson","line":"blue"}
{"station_id":40070,"station_name":"Jackson","line":"blue"}
{"station_id":40070,"station_name":"Jackson","line":"blue"}
{"station_id":41340,"station_name":"LaSalle","line":"blue"}
{"station_id":41340,"station_name":"LaSalle","line":"blue"}
{"station_id":41340,"station_name":"LaSalle","line":"blue"}
{"station_id":40430,"station_name":"Clinton","line":"blue"}
{"station_id":40430,"station_name":"Clinton","line":"blue"}
{"station_id":40470,"station_name":"Racine","line":"blue"}
Processed a total of 10 messages
root@41735495f605:/home/workspace/home#
```

Using the Schema Registry API, a schema is visible for arrivals and turnstile events.

Turnstile/Arrival schemas are correctly defined.

```
▼ root: {} 4 keys
  namespace: "com.udacity"
  type: "record"
  name: "turnstile.value"
▼ fields: [] 3 items
  ▼ 0: {} 2 keys
    name: "station_id"
    type: "int"
  ▼ 1: {} 2 keys
    name: "station_name"
    type: "string"
  ▼ 2: {} 2 keys
    name: "line"
    type: "string"

▼ root: {} 4 keys
  namespace: "com.udacity"
  type: "record"
  name: "arrival.value"
▼ fields: [] 7 items
  ▼ 0: {} 2 keys
    name: "station_id"
    type: "int"
```

- ▼ **1:** {} 2 keys
  - name:** "train\_id"
  - type:** "string"
- ▼ **2:** {} 2 keys
  - name:** "direction"
  - type:** "string"
- ▼ **3:** {} 2 keys
  - name:** "line"
  - type:** "string"
- ▼ **4:** {} 2 keys
  - name:** "train\_status"
  - type:** "string"
- ▼ **5:** {} 2 keys
  - name:** "prev\_station\_id"
  - ▶ **type:** [] 2 items
- ▼ **6:** {} 2 keys
  - name:** "prev\_direction"
  - ▶ **type:** [] 2 items

## Kafka Consumer

Stations, status, and weather data appear and update in the Transit Status UI.

You should use the given `offset_earliest` to determine the right offset for the consumer. You should not always set `earliest` for every consumer.

Tip: Even though we have manually set the offset for each partition, but we still need to set up the `config` for the initial consumer group or the consumer failed and fell over to another consumer.

All Blue, Green, and Red Line stations appear in the Transit Status UI.

Good job setting up the mechanism of consuming the messages

## Kafka REST Proxy

Using the kafka-console-consumer, weather messages are visible in the weather topic and are regularly produced as the simulation runs.

The weather messages can be properly consumed from the console.

```
root@41735495f605:/home/workspace/home# kafka-avro-console-consumer --topic org.chicago.cta.weather.v1 --from-beginning --bootstrap-server localhost:9092 --max-messages 10
{"temperature":63.552395,"status":"cloudy"}
{"temperature":72.77329,"status":"precipitation"}
{"temperature":70.64427,"status":"cloudy"}
{"temperature":72.55022,"status":"precipitation"}
{"temperature":74.0697,"status":"sunny"}
```

Using the Kafka Schema Registry REST API, a schema is defined for the weather topic.

Since we are using enum type for the weather status fields, you should make it as enum type in your schema instead of string type

Tip: For setting up enum type, it can prevent the field from being polluted by other unexpected values, which will help you prevent future cleanup and ease of downstream usage.

```
▼ root: {} 4 keys
  namespace: "com.udacity"
  name: "weather.value"
  type: "record"
  ▼ fields: [] 2 items
    ▼ 0: {} 2 keys
      name: "temperature"
      type: "float"
    ▼ 1: {} 2 keys
      name: "status"
      type: "string"
```

## Kafka Connect

Using the kafka-console-consumer, all stations defined in Postgres are visible in the stations topic.

All stations exist in the topic "postgres-stations"

```
root@41735495f605:/home/workspace/home# kafka-console-consumer --topic postgres-stations --from-beginning --bootstrap-server localhost:9092 --max-messages 10
{"stop_id":30001,"direction_id":"E","stop_name":"Austin (O'Hare-bound)","station_name":"Austin","station_descriptive_name":"Austin (Blue Line)","station_id":40010,"order":29,"red":false,"blue":true,"green":false}
{"stop_id":30002,"direction_id":"W","stop_name":"Austin (Forest Pk-bound)","station_name":"Austin","station_descriptive_name":"Austin (Blue Line)","station_id":40010,"order":29,"red":false,"blue":true,"green":false}
{"stop_id":30003,"direction_id":"E","stop_name":"Harlem (63rd-bound)","station_name":"Harlem/Lake","station_descriptive_name":"Harlem/Lake (Green Line)","station_id":40020,"order":0,"red":false,"blue":false,"green":true}
{"stop_id":30004,"direction_id":"W","stop_name":"Harlem (Terminal arrival)","station_name":"Harlem/Lake","station_descriptive_name":"Harlem/Lake (Green Line)","station_id":40020,"order":0,"red":false,"blue":false,"green":true}
{"stop_id":30005,"direction_id":"E","stop_name":"Pulaski (63rd-bound)","station_name":"Pulaski","station_descriptive_name":"Pulaski (Green Line)","station_id":40030,"order":7,"red":false,"blue":false,"green":true}
{"stop_id":30006,"direction_id":"W","stop_name":"Pulaski (Harlem-bound)","station_name":"Pulaski","station_descriptive_name":"Pulaski (Green Line)","station_id":40030,"order":7,"red":false,"blue":false,"green":true}
{"stop_id":30009,"direction_id":"W","stop_name":"Cicero (Harlem-bound)","station_name":"Cicero","station_descriptive_name":"Cicero (Green Line)","station_id":40480,"order":6,"red":false,"blue":false,"green":true}
{"stop_id":30012,"direction_id":"N","stop_name":"Belmont (O'Hare Branch) (O'Hare-bound)","station_name":"Belmont","station_descriptive_name":"Belmont (Blue Line)","station_id":40060,"order":8,"red":false,"blue":true,"green":false}
{"stop_id":30013,"direction_id":"S","stop_name":"Belmont (O'Hare Branch) (Forest Pk-bound)","station_name":"Belmont","station_descriptive_name":"Belmont (Blue Line)","station_id":40060,"order":8,"red":false,"blue":true,"green":false}
{"stop_id":30014,"direction_id":"N","stop_name":"Jackson/Dearborn (O'Hare-bound)","station_name":"Jackson","station_descriptive_name":"Jackson (Blue Line)","station_id":40070,"order":19,"red":false,"blue":true,"green":false}
Processed a total of 10 messages
root@41735495f605:/home/workspace/home#
```

Using the Kafka Connect REST API, the Kafka Connect configuration is configured to use JSON for both key and values.

Using the Schema Registry REST API, the schemas for stations key and value are visible.

Good job setting up config in the connector to the Postgres

Using the Kafka Connect REST API, the Kafka Connect configuration uses an incrementing ID, and the ID is configured to be `stop_id`.

Good job using `stop_id` as the incrementing ID

## Faust Streams

A consumer group for Faust is created on the Kafka Connect Stations topic.

Connect Stations topic is correctly created.

```
root@41735495f605:/home/workspace/home# python consumers/faust_stream.py tables
Tables
| name | help |
|-----|-----|
| org.chicago.cta.stations.table.v1 | Missing description: use Table(.., help="str") |

root@41735495f605:/home/workspace/home#
```

Data is ingested in the `Station` format and is then transformed into the `TransformedStation` format.

The faust program runs without any problems, well done.

```
root@41735495f605:/home/workspace/home# python consumers/faust_stream.py --json worker
rfapS+ v1.7.4
| id | stations-stream |
| transport | [URL('kafka://localhost:9092')] |
| store | memory |
| web | http://localhost:6066/ |
| log | -stderr- (warn) |
| pid | 1773 |
| hostname | 41735495f605 |
| platform | CPython 3.7.3 (Linux x86_64) |
| drivers | |
|   transport | aiokafka=1.0.6 |
|   web | aiohttp=3.6.2 |
| datadir | /home/workspace/home/stations-stream-data |
| appdir | /home/workspace/home/stations-stream-data/v1 |
starting> 😊
```

A topic is present in Kafka with the output topic name the student supplied. Inspecting messages in the topic, every station ID is represented.

Messages are correctly generated

```
root@41735495f605:/home/workspace/home# kafka-console-consumer --topic org.chicago.cta.stations.table.v1 --from-beginning --bootstrap-server localhost:9092 --max-messages 10
{"station_id": 40010, "station_name": "Austin", "order": 29, "line": "blue", "faust": {"ns": "consumers.faust_stream.TransformStation"}}
{"station_id": 40010, "station_name": "Austin", "order": 29, "line": "blue", "faust": {"ns": "consumers.faust_stream.TransformStation"}}
{"station_id": 40020, "station_name": "Harlem/Lake", "order": 0, "line": "green", "faust": {"ns": "consumers.faust_stream.TransformStation"}}
{"station_id": 40020, "station_name": "Harlem/Lake", "order": 0, "line": "green", "faust": {"ns": "consumers.faust_stream.TransformStation"}}
{"station_id": 40030, "station_name": "Pulaski", "order": 7, "line": "green", "faust": {"ns": "consumers.faust_stream.TransformStation"}}
{"station_id": 40030, "station_name": "Pulaski", "order": 7, "line": "green", "faust": {"ns": "consumers.faust_stream.TransformStation"}}
{"station_id": 40040, "station_name": "Cicero", "order": 6, "line": "green", "faust": {"ns": "consumers.faust_stream.TransformStation"}}
{"station_id": 40060, "station_name": "Belmont", "order": 8, "line": "blue", "faust": {"ns": "consumers.faust_stream.TransformStation"}}
{"station_id": 40060, "station_name": "Belmont", "order": 8, "line": "blue", "faust": {"ns": "consumers.faust_stream.TransformStation"}}
{"station_id": 40070, "station_name": "Jackson", "order": 19, "line": "blue", "faust": {"ns": "consumers.faust_stream.TransformStation"}}
Processed a total of 10 messages
root@41735495f605:/home/workspace/home#
```

## KSQL

Using the KSQL CLI, turnstile data is visible in the table `TURNSTILE`.

The turnstile data is visible on CLI.

```
ksql> select * from turnstile;
1621146496811 | 41330 | Montrose | blue
1621146496814 | 41330 | Montrose | blue
1621146496934 | 40570 | California | blue
1621146497012 | 41410 | Chicago | blue
1621146497319 | 40630 | Clark/Division | red
1621146497708 | 41270 | 43rd | green
1621146502756 | 40750 | Harlem | blue
1621146502761 | 40750 | Harlem | blue
1621146502762 | 40750 | Harlem | blue
1621146502820 | 40590 | Damen | blue
1621146502830 | 40590 | Damen | blue
1621146502883 | 40370 | Washington | blue
1621146502888 | 40370 | Washington | blue
1621146502895 | 40070 | Jackson | blue
1621146502948 | 40920 | Pulaski | blue
1621146502949 | 40920 | Pulaski | blue
1621146503076 | 41200 | Argyle | red
1621146503084 | 40770 | Lawrence | red
1621146503082 | 41200 | Argyle | red
```

Using the KSQL CLI, verify that station IDs have an associated `count` column.

The turnstile summary data is visible on CLI.

```
ksql> select * from turnstile_summary;
1621146525245 | 40700 | 40700 | 7
1621146525245 | 41430 | 41430 | 15
1621146525557 | 41430 | 41430 | 14
1621146525557 | 40290 | 40290 | 30
1621146531041 | 40510 | 40510 | 28
```

 RESUBMIT

 [DOWNLOAD PROJECT](#)



17

[CODE REVIEW COMMENTS](#)

## Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

[Watch Video \(3:01\)](#)

[RETURN TO PATH](#)

Rate this review

START

---