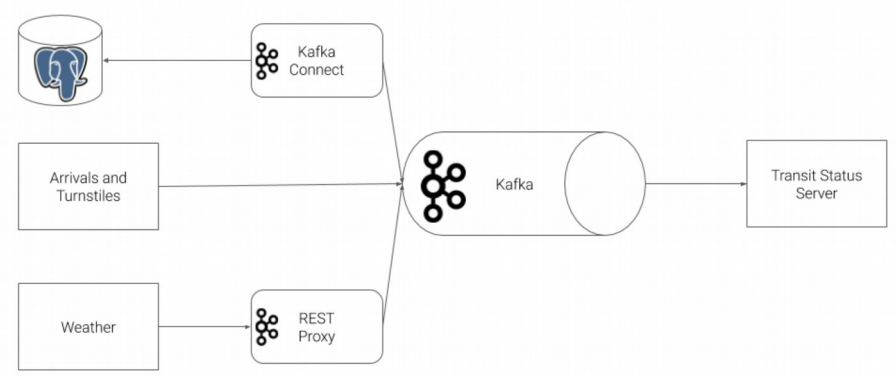
Starter Code and Data

Please find in the Resources tab, in the left sidebar of your classroom here, a zip file with all of the starter files and code referred to here in these directions. You can download those files and run your code locally, or you can use the Project Workspace we provide.

Project Directions

The Chicago Transit Authority (CTA) has asked us to develop a dashboard displaying system status for its commuters. We have decided to use Kafka and ecosystem tools like REST Proxy and Kafka Connect to accomplish this task.

Our architecture will look like so:



Step 1: Create Kafka Producers

The first step in our plan is to configure the train stations to emit some of the events that we need. The CTA has placed a sensor on each side of every train station that can be programmed to take an action whenever a train arrives at the station.

To accomplish this, you must complete the following tasks:

1. Complete the code in `producers/models/producer.py`.
2. Define a value schema for the arrival event in `producers/models/schemas/arrival_value.json` with the following attributes:

- station_id
- train_id
- direction
- line
- train_status
- prev_station_id
- prev_direction

3. Complete the code in `producers/models/station.py` so that:

- A topic is created for each station in Kafka to track the arrival events
- The station emits an arrival event to Kafka whenever the `Station.run()` function is called.
- Ensure that events emitted to Kafka are paired with the Avro key and value schemas

4. Define a value schema for the turnstile event in `producers/models/schemas/turnstile_value.json` with the following attributes

- station_id
- station_name
- line

5. Complete the code in `producers/models/turnstile.py` so that:

- A topic is created for each turnstile for each station in Kafka to track the turnstile events
- The station emits a turnstile event to Kafka whenever the `Turnstile.run()` function is called.
- Events emitted to Kafka are paired with the Avro key and value schemas

Step 2: Configure Kafka REST Proxy Producer

Our partners at the CTA have asked that we also send weather readings into Kafka from their weather hardware. Unfortunately, this hardware is old and we cannot use the Python Client Library due to hardware restrictions. Instead, we are going to use HTTP REST to send the data to Kafka from the hardware using Kafka's REST Proxy.

To accomplish this, you must complete the following tasks:

1. Define a value schema for the weather event

   in `producers/models/schemas/weather_value.json` with the following attributes:

- temperature
- status

2. Complete the code in `producers/models/weather.py` so that:

- A topic is created for weather events
- The weather model emits weather event to Kafka REST Proxy whenever the Weather.run() function is called.
- NOTE: When sending HTTP requests to Kafka REST Proxy, be careful to include the correct Content-Type. Pay close attention to the **examples in the documentation** for more information.
- Events emitted to REST Proxy are paired with the Avro key and value schemas

Step 3: Configure Kafka Connect

Finally, we need to extract station information from our PostgreSQL database into Kafka. We've decided to use the **Kafka JDBC Source Connector**.

To accomplish this, you must complete the following tasks:

1. Complete the code and configuration in producers/connectors.py

- Please refer to the **Kafka Connect JDBC Source Connector Configuration Options** for documentation on the options you must complete.
- You can run this file directly to test your connector, rather than running the entire simulation.
- Make sure to use the Kafka Connect API, kafka-console-consumer, and kafka-topics CLI tool to check the status and output of the Connector

- To delete a misconfigured connector: CURL -X DELETE localhost:8083/connectors/stations

Step 4: Configure the Faust Stream Processor

We will leverage Faust Stream Processing to transform the raw Stations table that we ingested from Kafka Connect. The raw format from the database has more data than we need, and the line color information is not conveniently configured. To remediate this, we're going to ingest data from our Kafka Connect topic, and transform the data.

To accomplish this, you must complete the following tasks:

- Complete the code and configuration in `consumers/faust_stream.py`

  Watch Out! You must run this Faust processing application with the following command:
  `faust -A faust_stream worker -l info`

Step 5: Configure the KSQL Table

Next, we will use KSQL to aggregate turnstile data for each of our stations. Recall that when we produced turnstile data, we simply emitted an event, not a count. What would make this data more useful would be to summarize it by station so that downstream applications always have an up-to-date count.

To accomplish this, you must complete the following tasks:

- Complete the queries in `consumers/ksql.py`.

  Tips
- The KSQL CLI is the best place to build your queries. Try `ksql` in your workspace to enter the CLI.
- You can run this file on its own simply by running `python ksql.py`.
- Made a mistake in table creation? `DROP TABLE <your_table>`. If the CLI asks you to terminate a running query, you can `TERMINATE <query_name>`.

Step 6: Create Kafka Consumers

With all of the data in Kafka, our final task is to consume the data in the web server that is going to serve the transit status pages to our commuters.

To accomplish this, you must complete the following tasks:

1. Complete the code in `consumers/consumer.py`
2. Complete the code in `consumers/models/line.py`
3. Complete the code in `consumers/models/weather.py`
4. Complete the code in `consumers/models/station.py`

Other Resources and Documentation

In addition to the course content you have already reviewed, you may find the following examples and documentation helpful in completing this project:

- **Confluent Python Client Documentation**
- **Confluent Python Client Usage and Examples**
- **REST Proxy API Reference**
- **Kafka Connect JDBC Source Connector Configuration Options**

Directory Layout

The project consists of two main directories, `producers` and `consumers`.
The following directory layout indicates with a * the files that the student is responsible for modifying. Instructions for what is required are present as comments in each file.

```
* - Indicates that the student must complete the code in this file

├── consumers
│   ├── consumer.py *
│   ├── models
│   │   ├── lines.py
│   │   ├── line.py *
│   │   ├── station.py *
│   │   └── weather.py *
│   ├── requirements.txt
│   ├── server.py
│   └── templates
│       └── status.html
└── producers
    ├── connector.py *
    ├── models
    │   ├── line.py
    │   ├── producer.py *
    │   ├── schemas
    │   │   ├── arrival_key.json
    │   │   ├── arrival_value.json *
    │   │   ├── turnstile_key.json
    │   │   ├── turnstile_value.json *
    │   │   ├── weather_key.json
    │   │   └── weather_value.json *
    │   ├── station.py *
    │   ├── train.py
    │   ├── turnstile.py *
    │   ├── turnstile_hardware.py
    │   └── weather.py *
    ├── requirements.txt
    └── simulation.py
```

Running and Testing

Running in the Classroom Project Workspace (recommended)

The Project Workspace environment is already preconfigured with all of the services you need to complete your project. No additional configuration is required.

The following services are available in the classroom workspace environment:

Service	Host URL	Docker URL	Username	Password
Public Transit Status	http://localhost:8888	n/a		
Kafka	PLAINTEXT://localhost:9092	PLAINTEXT://kafka0:9092		
REST Proxy	http://localhost:8082	http://rest-proxy:8082/		
Schema Registry	http://localhost:8081	http://schema-registry:8081/		
Kafka Connect	http://localhost:8083	http://kafka-connect:8083		
KSQL	http://localhost:8088	http://ksql:8088		
PostgreSQL	jdbc:postgresql://localhost:5432/cta	jdbc:postgresql://postgres:5432/cta	cta_admin	chicago

Running on Your Computer

NOTE: You must allocate *at least* 4gb RAM to your Docker-Compose environment to run locally.

To run the simulation, you must first start up the Kafka ecosystem on your machine utilizing Docker-Compose.

```
%> docker-compose up
```

Docker-Compose will take 3-5 minutes to start, depending on your hardware. Please be patient and wait for the Docker-Compose logs to slow down or stop before beginning the simulation.

Once Docker-Compose is ready, the following services will be available on your local machine:

Service	Host URL	Docker URL	Username	Password
Public Transit Status	http://localhost:8888	n/a		
Kafka	PLAINTEXT://localhost:9092,PLAINTEXT://localhost:9093,PLAINTEXT://localhost:9094	PLAINTEXT://kafka0:9092,PLAINTEXT://kafka1:9093,PLAINTEXT://kafka2:9094		
REST Proxy	http://localhost:8082	http://rest-proxy:8082/		
Schema Registry	http://localhost:8081	http://schema-registry:8081/		
Kafka Connect	http://localhost:8083	http://kafka-connect:8083		
KSQL	http://localhost:8088	http://ksql:8088		
PostgreSQL	jdbc:postgresql://localhost:5432/cta	jdbc:postgresql://postgres:5432/cta	cta_admin	chicago

Note that to access these services from your own machine, you will always use the `Host URL` column.
When configuring services that run within Docker-Compose, like Kafka Connect, **you must use the `Docker URL`**. When you configure the JDBC Source Kafka Connector, for example, you will want to use the value from the `Docker URL` column.

Running the Simulation

There are two pieces to the simulation, the `producer` and `consumer`. As you develop each piece of the code, it is recommended that you only run one piece of the project at a time.
However, when you are ready to verify the end-to-end system prior to submission, it is critical that you open a terminal window for each piece and run them at the same time. **If you do not run both the `producer` and `consumer` at the same time you will not be able to successfully complete the project.**
To run the `producer`:
If using Project Workspace:

1. `cd producers`
2. `python simulation.py`

If using your computer:

1. `cd producers`
2. `virtualenv venv`
3. `. venv/bin/activate`
4. `pip install -r requirements.txt`
5. `python simulation.py`

Once the simulation is running, you may hit `Ctrl+C` at any time to exit.
To run the Faust Stream Processing Application:
If using Project Workspace:

1. `cd consumers`
2. `faust -A faust_stream worker -l info`

If using your computer:

1. `cd consumers`

2. `virtualenv venv`
3. `. venv/bin/activate`
4. `pip install -r requirements.txt`
5. `faust -A faust_stream worker -l info`

To run the KSQL Creation Script:

If using Project Workspace:

1. `cd consumers`
2. `python ksql.py`

If using your computer:

1. `cd consumers`
2. `virtualenv venv`
3. `. venv/bin/activate`
4. `pip install -r requirements.txt`
5. `python ksql.py`

To run the `consumer`: (**NOTE**: Do not run the consumer until you have reached Step 6!)
If using Project Workspace:

1. `cd consumers`
2. `python server.py`

If using your computer:

1. `cd consumers`
2. `virtualenv venv`
3. `. venv/bin/activate`
4. `pip install -r requirements.txt`
5. `python server.py`

Once the server is running, you may hit `Ctrl+C` at any time to exit.

Project Submission

Please check your work against the project **rubric** before you submit it. Your reviewer will be using this rubric to assess your project work.

The only thing you need to submit for this project is your code. To do this, you have two options:

- You may submit a link to your GitHub repo on the project page: "Optimizing Public Transportation."
- Or for those of you developing your code in the classroom workspace, you may submit your workspace code directly.