**Project Overview: Kafka-Powered Transit Status Tracker**

In this project, we're going to build a **transit status tracker** for the **city of Chicago** using **Apache Kafka and its ecosystem**.

As you can see on the screen, we're going to go ahead and create a **website** that looks like this. It's going to be powered by **real-time event streaming data** from **Kafka, Faust, KSQL,** as well as some of the other tools you learned about in the class.

To complete this project, we've decided that we're going to do **a few things**. This is the **architecture** that we're going to have once we're complete with the project. You'll notice that **Kafka is a central component**.

**Key Steps in the Project**

1. **Create a Kafka Producer**
   * This producer will generate **train arrival** and **turnstile information** into our Kafka cluster.
   * **Arrivals:** Indicate when a train has arrived at a particular station.
   * **Turnstile Events:** Indicate when a passenger has entered the station.
2. **Build a Kafka REST Proxy Producer**
   * This will be a **Python script** that periodically emits **weather data** via REST Proxy and places it into Kafka.
3. **Develop a Kafka Connect JDBC Source Connector**
   * This will connect to **PostgreSQL**, extract data from the **stations table**, and place it into Kafka.
4. **Consume Data from Kafka Topics**
   * We will use a **Kafka consumer** to **consume data** from these topics.
   * **Faust and KSQL** will be used to extract and transform this data.

**Recommended Workflow**

* The **first step** is to create the **Kafka producers**.
* The **README** contains specific tasks to complete in the project.
* After setting up the producers, proceed with:
  + **Kafka REST Proxy Producer**
  + **Kafka Connect JDBC Source Connector**
  + **Consuming Data** from these sources
  + **Building a real-time dashboard** for visualization

**Project Overview: Kafka-Powered Transit Status Tracker (Part 2)**

Once you're ready to actually start the project, I recommend going to the **producer's folder** first.

In the **producer's folder**, you'll find the files that you need to complete:

* **connector.py** → for Kafka Connect
* **Models directory** → contains producer code

Now, in the **models directory**, we're going to take a look first at the **producer code**.

**Producer Code**

* The producer file contains **comments** guiding you on what to complete.
* You need to **fill out broker properties**, configure an **Avro producer**, create a **Kafka topic**, and handle **closing the producer**.
* Some **wrapping code** is already provided, but you will need to **fill in the blanks**.
* This Kafka producer will be used by **all the other classes** that interact with Kafka.

**Station Data Producer**

* The **station class** inherits from the producer.
* Once the producer is complete, it will **interact with Kafka automatically**.
* You need to:
  + Provide **topic name, value schema, number of partitions, replicas**.
  + Implement the **run function** to produce data.

**Turnstile Data Producer**

* The **turnstile class** follows a **similar process**.
* You will define a **value schema** for the turnstile events.

**Avro Schemas**

* The project requires you to **define Avro schemas**.
* In the project folder, look for the **"schemas"** directory.
* There are **multiple Avro schemas** that must be completed before the project can function correctly.
* Example: If you're working on **turnstile data**, open the **turnstile value schema** file.
* Edit these files using a **JSON editor** to ensure correct formatting.

**Next Steps**

* Complete the **Kafka producers** before proceeding.
* Define all **Avro schemas** correctly.
* Test **station and turnstile producers** to ensure they send data to Kafka.

**Project Overview: Kafka-Powered Transit Status Tracker (Part 3)**

With that done, we're ready to move on to the **consumer**.  
Let's move back to the **root folder** and navigate to **consumers**.

The first file we will look at is **solver.py**.

**Understanding solver.py**

* This file **does not require any modifications**.
* It **instantiates Kafka consumers** and sets up a **web application**.
* The web application listens on **port 8888**, where you can access the web project.

**Faust Stream Processing**

* The **Faust stream** is already set up with input and output stations.
* The goal is to **filter the station data**, keeping only specific fields.
* The output should be a **transformed station** with the necessary fields.
* **Instructions are provided in the comments** to guide completion.

**KSQL Stream Processing**

* KSQL will be used to transform **turnstile data** into a **turnstile summary**.
* The **skeleton of a KSQL statement** is provided for you to complete.
* Instead of using the **KSQL CLI**, we will **post KSQL queries via requests**.
* You need to **fill in the missing parts of the KSQL statement**.

**Running the Project**

* You **must run ksql.py and fauststream.py manually**.
* These scripts **will not start automatically** when running the server.
* They are meant to be **developed independently** from the server.

**Next Steps**

* Complete the **Faust stream filtering**.
* Fill in the **KSQL transformation** query.
* Ensure **Kafka consumers** process data correctly.
* Run **ksql.py and fauststream.py** manually for proper functionality.

**Project Overview: Kafka-Powered Transit Status Tracker (Part 4)**

The final step of the project is to navigate to the **Consumer's Models folder** and complete the following files:

* **line.py**
* **weather.py**

**Completing process\_message**

* The key function you need to complete is **process\_message**.
* Right now, there are placeholder conditions (if true, true, true).
* You must **properly handle messages** and determine how to **route them**.

**Processing Weather Data (weather.py)**

* You need to **extract temperature and weather status** from incoming Kafka messages.
* Update the **object representation** accordingly.

**Kafka Consumer Implementation (consumer.py)**

* **consumer.py** must be completed **before** working on the models.
* This file is similar to the **producer model** from the earlier part of the project.
* Key tasks:
  + Set up a **Kafka consumer**.
  + Handle **different data formats** (Avro or JSON).
  + Subscribe to **Kafka topics**.
  + Implement the **on\_assign** function:
    - Remember that sometimes we need to **rewind partitions** to the **first known offset**.
  + Complete the **consume** and **close** functions.

**Running the Final Project**

* Once all steps are completed, you can **run the Python server**.
* Expect an **output that looks like the project visualization**.
* **It may take a few minutes to start**:
  + If you see **duplicate trains**, wait for the system to stabilize.
  + Due to limited resources and the **speed of simulation**, it can take **a couple of minutes** to function correctly.
  + **Look for updates** in:
    - **Temperature readings**
    - **Total turnstile entries**
    - **Trains moving from station to station**

**Final Steps**

Now you're ready to get started with the project!  
This project involves **real-time event streaming**, so be patient while everything initializes.