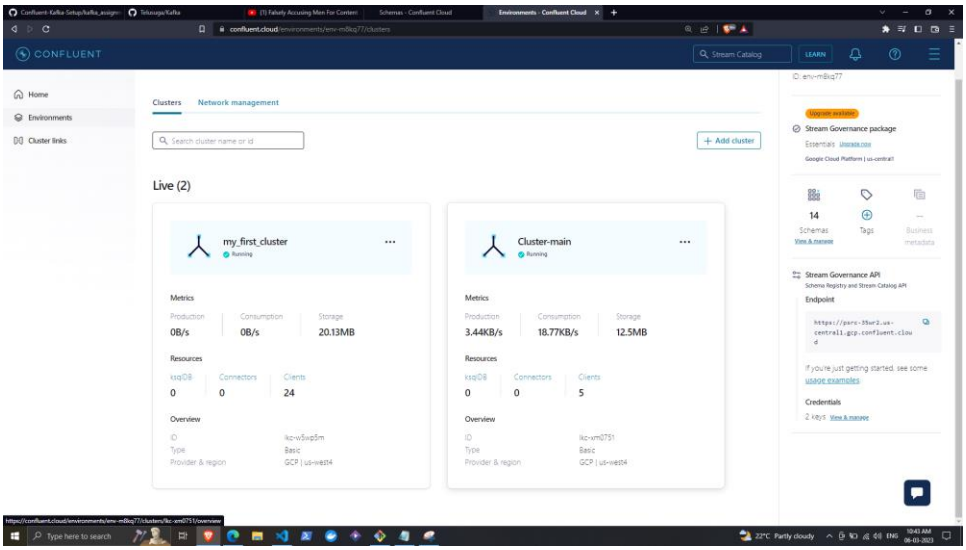
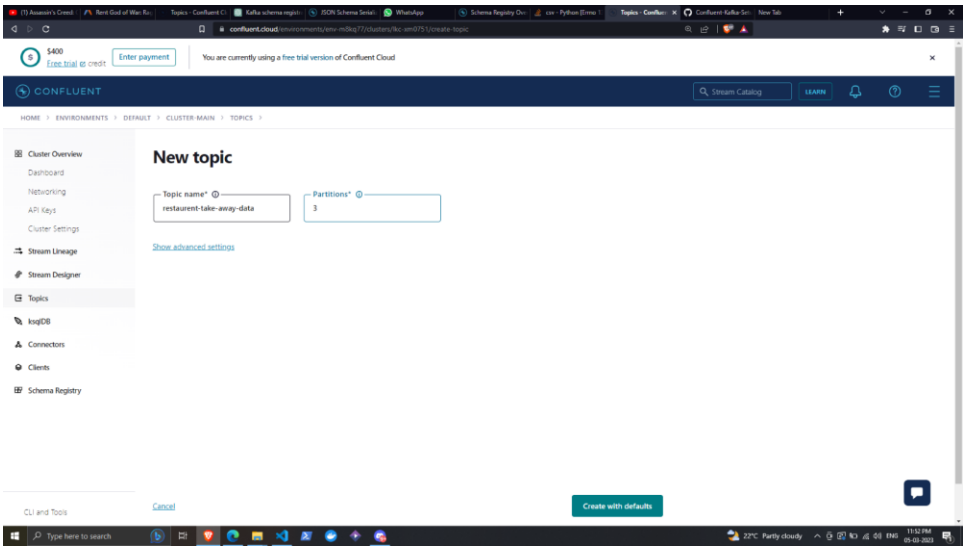
1. Setup Confluent Kafka Account



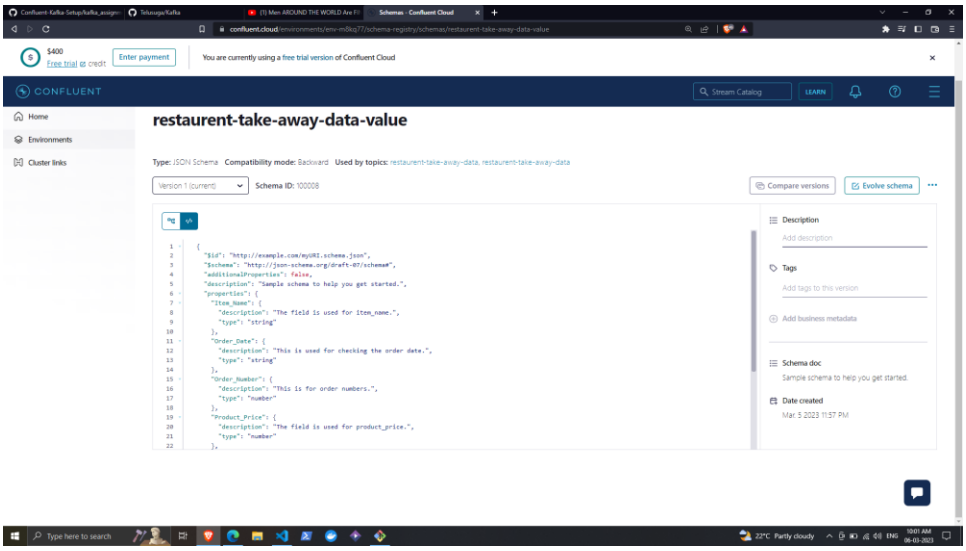
Created a cluster using GCP

Cluster:Cluster-main

2)Create one kafka topic named as "restaurent-take-away-data" with 3 partitions



3)Setup key (string) & value (json) schema in the confluent schema registry



4)Write a kafka producer program (python or any other language) to read data records from restaurent data csv file, make sure schema is not hardcoded in the producer code, read the latest version of schema and schema_str from schema registry and use it for data serialization.

Producer Code:

```
import argparse

from uuid import uuid4

from six.moves import input

from confluent_kafka import Producer

from confluent_kafka.serialization import StringSerializer, SerializationContext,
MessageField

from confluent_kafka.schema_registry import SchemaRegistryClient

from confluent_kafka.schema_registry.json_schema import JSONSerializer

# from confluent_kafka.schema_registry import *

import pandas as pd

from typing import List
```

```
FILE_PATH = 'C:/Users/javva/Downloads/BigData/Kafka-mini/Restaurant_data.csv'
columns = ['Order_Number', 'Order_Date', 'Item_Name',
           'Quantity', 'Product_Price', 'Total_products']


API_KEY = 'TVM3PGDPSLYOQY3T'
ENDPOINT_SCHEMA_URL = 'https://psrc-35wr2.us-central1.gcp.confluent.cloud'
API_SECRET_KEY =
'8+QpSlOX/xtaBKbDbvWgZ54dug2eJ6VDifZFjzM8F+XijF+GRdqB419GK99ORpvy'
BOOTSTRAP_SERVER = 'pkc-6ojv2.us-west4.gcp.confluent.cloud:9092'
SECURITY_PROTOCOL = 'SASL_SSL'
SSL_MACHENISM = 'PLAIN'
SCHEMA_REGISTRY_API_KEY = 'XNL5OLFHUFODSM44'
SCHEMA_REGISTRY_API_SECRET =
'RkXRkfIWUFJ1W7Wkcgs0xY0Dzu2ZCji13M7Zr+6Q02A/H5rZfapemi3emg1wukAM'



def sasl_conf():
    sasl_conf = {'sasl.mechanism': SSL_MACHENISM,
                 # Set to SASL_SSL to enable TLS support.
                 #  'security.protocol': 'SASL_PLAINTEXT'}
                 'bootstrap.servers': BOOTSTRAP_SERVER,
                 'security.protocol': SECURITY_PROTOCOL,
                 'sasl.username': API_KEY,
                 'sasl.password': API_SECRET_KEY
                 }
    return sasl_conf
```

```
def schema_config():
    return {'url': ENDPOINT_SCHEMA_URL,


            'basic.auth.user.info':
f"{SCHEMA_REGISTRY_API_KEY}:{SCHEMA_REGISTRY_API_SECRET}"


            }



class Car:
    def __init__(self, record: dict):
        for k, v in record.items():
            setattr(self, k, v)


        self.record = record


    @staticmethod
    def dict_to_car(data: dict, ctx):
        return Car(record=data)


    def __str__(self):
        return f"{self.record}"
```

```
def get_car_instance(file_path):
    df = pd.read_csv(file_path)
    df = df.iloc[:, :]
    cars: List[Car] = []
    for data in df.values:
        car = Car(dict(zip(columns, data)))
        cars.append(car)
        yield car



def car_to_dict(car: Car, ctx):
    """
    Returns a dict representation of a User instance for serialization.
    Args:
        user (User): User instance.
        ctx (SerializationContext): Metadata pertaining to the serialization
            operation.
    Returns:
        dict: Dict populated with user attributes to be serialized.
    """

    # User._address must not be serialized; omit from dict
    return car.record



def delivery_report(err, msg):
```

```
    """
    Reports the success or failure of a message delivery.
    Args:
        err (KafkaError): The error that occurred on None on success.
        msg (Message): The message that was produced or failed.
    """

    if err is not None:
        print("Delivery failed for User record {}: {}".format(msg.key(), err))
        return
    print('User record {} successfully produced to {} [{}] at offset {}'.format(
        msg.key(), msg.topic(), msg.partition(), msg.offset()))


def main(topic):
    schema_registry_conf = schema_config()
    schema_registry_client = SchemaRegistryClient(schema_registry_conf)
    schema_subject = 'restaurent-take-away-data-value'
    schema_check = schema_registry_client.get_latest_version(schema_subject)
    schema_str = schema_check.schema.schema_str

    string_serializer = StringSerializer('utf_8')
    json_serializer = JSONSerializer(
        schema_str, schema_registry_client, car_to_dict)

    producer = Producer(sasl_conf())
```

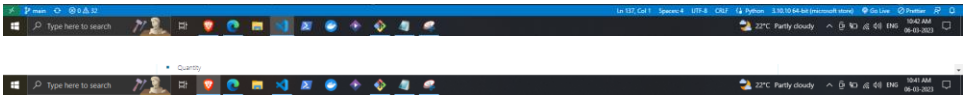
```
    print("Producing user records to topic {}. ^C to exit.".format(topic))
    # while True:
    # Serve on_delivery callbacks from previous calls to produce()
    producer.poll(0.0)
    try:
        for car in get_car_instance(file_path=FILE_PATH):
            print(car)
            producer.produce(topic=topic,
                             key=string_serializer(str(uuid4()), car_to_dict),
                             value=json_serializer(
                                 car, SerializationContext(topic,
MessageField.VALUE)),
                             on_delivery=delivery_report)
    except KeyboardInterrupt:
        pass
    except ValueError:
        print("Invalid input, discarding record...")
        pass


    print("\nFlushing records...")
    producer.flush()



main("restaurent-take-away-data")
```

6)From producer code, publish data in Kafka Topic one by one and use dynamic key while publishing the records into the Kafka Topic



7)Write kafka consumer code and create two copies of same consumer code and save it with different names (kafka_consumer_1.py & kafka_consumer_2.py),

again make sure lates schema version and schema_str is not hardcoded in the consumer code, read it automatically from the schema registry to desrialize the data.

Now test two scenarios with your consumer code:

a.) Use "group.id" property in consumer config for both consumers and mention different group_ids in kafka_consumer_1.py & kafka_consumer_2.py,

apply "earliest" offset property in both consumers and run these two consumers from two different terminals. Calculate how many records each consumer

consumed and printed on the terminal

b.) Use "group.id" property in consumer config for both consumers and mention same group_ids in kafka_consumer_1.py & kafka_consumer_2.py,

apply "earliest" offset property in both consumers and run these two consumers from two different terminals. Calculate how many records each consumer

consumed and printed on the terminal

Note:

1)While using same group_id the consumers pulled the same message.

2)While using different group_id the consumer_1 pulled different message where as the consumer_2 in which group_id has been changed to group2 pulled earlier message where group_id was same.

8)Once above questions are done, write another kafka consumer to read data from kafka topic and from the consumer code create one csv file "output.csv” and append consumed records output.csv file

Created a consumer file which will push a messages to a output csv file