Avro producer with python

Create Avro Producers for Confluent Kafka with Python

Mrugank Ray · Follow

5 min read · Feb 21

3

Overview:

This article will teach you how to create an Avro producer using the Confluent kafka library in python.

Setup:

Clone Big Data Cluster repo. This will set up an environment for producing messages in Avro format for Kafka.
Run the following command to initialize a docker cluster

```
sudo docker-compose -f kafka-zookeper.yaml up
```

NOTE: You need docker on your computer to run the following commands. If you already have a kafka cluster running, you can skip this step.

Dependency:

Running avro producer will require some dependencies to be installed on your system. In our project directory, let's create a **requirements.txt** file and paste the following text in there.

```
confluent-kafka==1.9.2
fastavro==1.4.7
requests==2.27.1
```

avro_producer_requirements.txt hosted with ❤ by GitHub view raw

requirements.txt

Let's install the dependencies using the following command in the project directory

```
pip install -r requirements.txt
```

Source:

We're going to use Wikimedia's event streams as the source. To learn more visit https://stream.wikimedia.org/?doc#/streams/get_v2_stream_recentchange. We use this source because it's publicly available and easy to use.
For reading event streams, we will use **requests** library available for python. Requesting https://stream.wikimedia.org/v2/stream/recentchange gives you events that look like this.

wikimedia event stream

Get data from stream

Let's take a look at how to get data from streams.

```python
import requests
import json

init_string = 'data: '
source_url = 'https://stream.wikimedia.org/v2/stream/recentchange'

def avro_producer(source_url):
    s = requests.Session()

    with s.get(source_url, headers=None, stream=True) as resp:
        for line in resp.iter_lines():
            if line:
                decoded_line = line.decode()
                if decoded_line.find(init_string) >= 0:
                    # remove data: to create a valid json
                    decoded_line = decoded_line.replace(init_string, "")
                    # convert to json
                    decoded_json = json.loads(decoded_line)
                    print(decoded_json)

avro_producer(source_url)
```

get_wikimedia_data.py hosted with ❤ by GitHub view raw

avro_producer.py

Let's understand the code:

- Line 8 to 10: We create a session object and establish a TCP connection.
- Line 11 to 13: We loop through the events and then we need to decode the byte string to UTF-8.
- Line 14: Each event data is stored as a string. We search for events that have a "data: "string since we're interested in these.
- Line 16 to 19: We remove the string "data: " and convert it into a valid JSON object.

Create kafka topic:

Let's create a Kafka topic. To do this, we need to log into our "kafka-broker" container. Once logged in run the following command:

```
kafka-topics - bootstrap-server kafka-broker:29092 - create - topic wikimedia -
```

This creates a topic named “wikimedia” with one partition and one replication factor.
NOTE: The kafka cluster only has one broker, so set the replication factor to 1.

Avro schema:

In this section, we will create the avro schema and register it in schema registry.

Define the Avro schema

Schemas can be created for both key and value, but we’ll only create a schema for value here. The value is the data that’s streaming from Wikimedia.
This site helps you create AVRO schema from JSON.
Here is schema that I’ve created.

Now let’s make a **constants.py** file where we define constants like SCHEMA_STR and assign stringified schema.

```
1  SCHEMA_STR="{\"type\":\"record\",\"name\":\"value_wikimedia\",\"namespace\":\"wikimedia\",\"field
```

wikimedia_stringified_schema.py hosted with ❤ by GitHub view raw

constants.py

Check out this site for more info on AVRO schema.

Add schema to the registry

There are multiple ways to add schema to the schema registry, like **REST APIs** and **Confluent Kafka library**.

In this article, we're going to use the Schema Registry Client.

Let's take a look at how to register a schema

```python
# 3rd party library imported
from confluent_kafka.schema_registry import SchemaRegistryClient
from confluent_kafka.schema_registry import Schema

# imort from constants
from constants import SCHEMA_STR

schema_registry_url = 'http://localhost:8083'
kafka_topic = 'test-blog'
schema_registry_subject = f"{kafka_topic}-value"

def get_schema_from_schema_registry(schema_registry_url, schema_registry_subject):
    sr = SchemaRegistryClient({'url': schema_registry_url})
    latest_version = sr.get_latest_version(schema_registry_subject)

    return sr, latest_version

def register_schema(schema_registry_url, schema_registry_subject, schema_str):
    sr = SchemaRegistryClient({'url': schema_registry_url})
    schema = Schema(schema_str, schema_type="AVRO")
    schema_id = sr.register_schema(subject_name=schema_registry_subject, schema=schema)

    return schema_id

def update_schema(schema_registry_url, schema_registry_subject, schema_str):
    sr = SchemaRegistryClient({'url': schema_registry_url})
    versions_deleted_list = sr.delete_subject(schema_registry_subject)
    print(f"versions of schema deleted list: {versions_deleted_list}")

    schema_id = register_schema(schema_registry_url, schema_registry_subject, schema_str)
    return schema_id

schema_id = register_schema(schema_registry_url, schema_registry_subject, SCHEMA_STR)
print(schema_id)

sr, latest_version = get_schema_from_schema_registry(schema_registry_url, schema_registry_subjec
print(latest_version.schema.schema_str)
```

register_schema.py hosted with ❤ by GitHub view raw

avro_producer.py

Let's understand the code:

- Line 9: We set the kafka topic name.
- Line 10: The topic name is suffixed with "-value" for a value schema. If your schema is a key schema, it's suffixed with "-key".
- Line 12 to 16: We create a schema registry client instance and get the latest version of the schema using the subject. You can learn more about it here
- Line 18 to 23: We pass the schema_type of AVRO along with the stringified schema data to create the Schema object. Then we register the schema in the registry, which returns the schema ID.
- Line 25 to 31: There's no update function in confluent kafka library, so we create one function that deletes the schema and registers the updated schema.
- Line 33: We register the schema.
- Line 36: We get the latest version schema's details.

Avro producer:

Let's get started. We will break down the steps as we go along.

Send avro data to kafka

We'll update the avro_producer function to send data.

```python
import requests
import json

# 3rd party library imported
from confluent_kafka import SerializingProducer
from confluent_kafka.schema_registry import SchemaRegistryClient
from confluent_kafka.schema_registry.avro import AvroSerializer
from confluent_kafka.schema_registry import Schema

# imort from constants
from constants import SCHEMA_STR

init_string = 'data: '
source_url = 'https://stream.wikimedia.org/v2/stream/recentchange'
kafka_url = 'localhost:9092'
schema_registry_url = 'http://localhost:8083'
kafka_topic = 'wikimedia'
schema_registry_subject = f"{kafka_topic}-value"

def delivery_report(errmsg, msg):
    if errmsg is not None:
        print("Delivery failed for Message: {} : {}".format(msg.key(), errmsg))
        return
    print('Message: {} successfully produced to Topic: {} Partition: [{}] at offset {}'.format(m

def avro_producer(source_url, kafka_url, schema_registry_url, schema_registry_subject):
    # schema registry
    sr, latest_version = get_schema_from_schema_registry(schema_registry_url, schema_registry_su


    value_avro_serializer = AvroSerializer(schema_registry_client = sr,
                                           schema_str = latest_version.schema.schema_str,
                                           conf={
                                               'auto.register.schemas': False
                                             }
                                           )

    # Kafka Producer
    producer = SerializingProducer({
        'bootstrap.servers': kafka_url,
        'security.protocol': 'plaintext',
        'value.serializer': value_avro_serializer,
        'delivery.timeout.ms': 120000, # set it to 2 mins
        'enable.idempotence': 'true'
    })
```

```
    });

    s = requests.Session()

    with s.get(source_url, headers=None, stream=True) as resp:
        for line in resp.iter_lines():
            if line:
                decoded_line = line.decode()
                if decoded_line.find(init_string) >= 0:
                    # remove data: to create a valid json
                    decoded_line = decoded_line.replace(init_string, "")
                    # convert to json
                    decoded_json = json.loads(decoded_line)

                    try:
                        # print(decoded_line + '\n')
                        producer.produce(topic=kafka_topic, value=decoded_json, on_delivery=deli

                        # Trigger any available delivery report callbacks from previous produce(
                        events_processed = producer.poll(1)
                        print(f"events_processed: {events_processed}")

                        messages_in_queue = producer.flush(1)
                        print(f"messages_in_queue: {messages_in_queue}")
                    except Exception as e:
                        print(e)

def get_schema_from_schema_registry(schema_registry_url, schema_registry_subject):
    sr = SchemaRegistryClient({'url': schema_registry_url})
    latest_version = sr.get_latest_version(schema_registry_subject)

    return sr, latest_version

def register_schema(schema_registry_url, schema_registry_subject, schema_str):
    sr = SchemaRegistryClient({'url': schema_registry_url})
    schema = Schema(schema_str, schema_type="AVRO")
    schema_id = sr.register_schema(subject_name=schema_registry_subject, schema=schema)

    return schema_id

def update_schema(schema_registry_url, schema_registry_subject, schema_str):
    sr = SchemaRegistryClient({'url': schema_registry_url})
    versions_deleted_list = sr.delete_subject(schema_registry_subject)
    print(f"versions of schema deleted list: {versions_deleted_list}")

```

```
90        schema_id = register_schema(schema_registry_url, schema_registry_subject, schema_str)
91        return schema_id
92
93    avro_producer(source_url, kafka_url, schema_registry_url, schema_registry_subject)
```

avro_producer.py hosted with ❤ by GitHub view raw

Let's understand the code:

- Line 20 to 24: It's a callback function that's called when the message is delivered successfully or failed. In case the message doesn't get delivered, we log the error to console. In case the message gets delivered, we log the key, topic, partition and offset.
- Line 28 to 36: Once we get the latest schema, we create an avro serializer and make "auto.register.schemas" False in the config so that the library won't create a new schema.
- Line 39 to 45: We set up a producer and some configurations. You can read about SerializingProducer here. "delivery.timeout.ms" sets the maximum time a producer can take to deliver a message (including retries). We have set it to 120000 milliseconds or 2 mins. When we set "enable.idempotence" to true, messages are produced exactly once and in the original order. You can read more about config here.
- Line 61: This function is async in nature and calls the delivery_report function once messages have been delivered or failed.
- Line 64: This polls the producer for events and gets back the number of events processed. We can optionally pass max waiting time.
- Line 67: This function waits for all messages in the Producer queue to be delivered. It returns the number of messages still in the queue. We can optionally pass a waiting time. If messages are delivered within the waiting time then it would return zero.

Output

```
Message: None successfully produced to Topic: wikimedia Partition: [0] at offset 579
events_processed: 1
messages_in_queue: 0
Message: None successfully produced to Topic: wikimedia Partition: [0] at offset 580
events_processed: 1
messages_in_queue: 0
```

Avro producer

Avro Consumer:

We will discuss two ways to consume the messages.

CLI

We will use CLI to consume the messages from Kafka topic. To do this, we need to log into our “schema-registry” container. Once logged in run the following command:

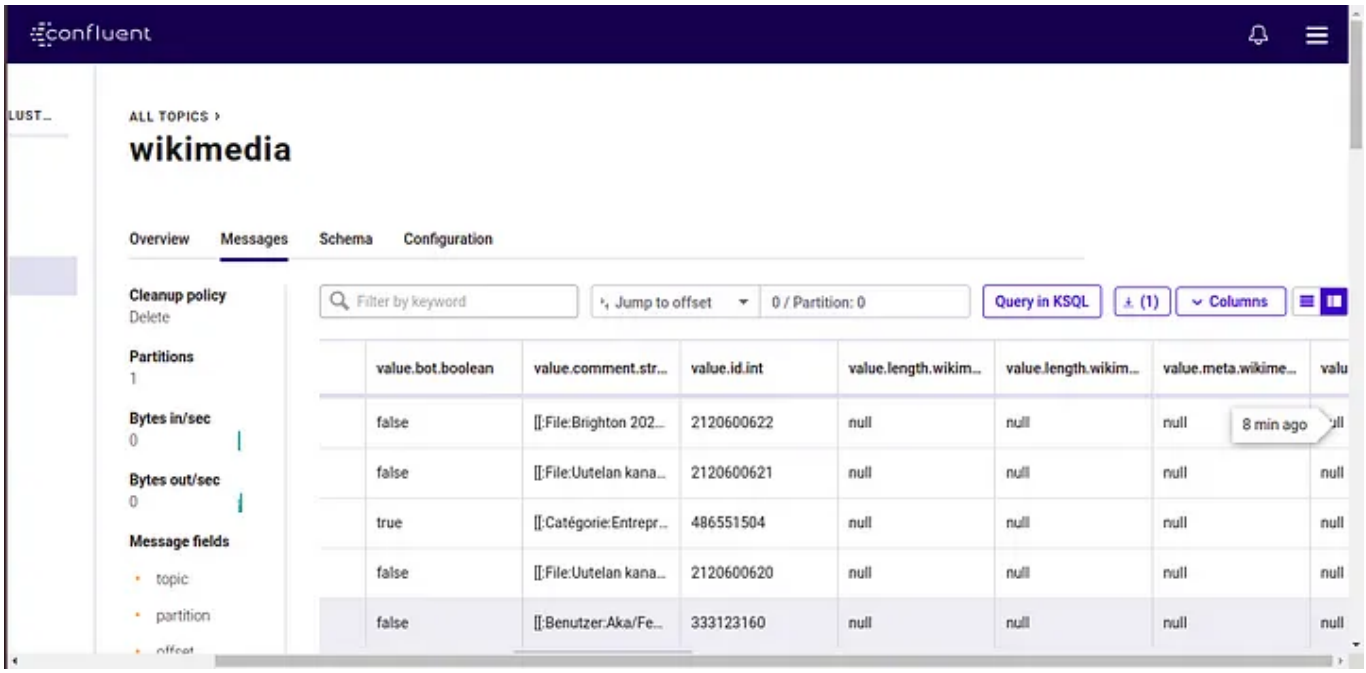
```
kafka-avro-console-consumer \
 - bootstrap-server kafka-broker:29092 \
 - topic wikimedia \
 - from-beginning \
 - property schema.registry.url=http://schema-registry:8083
```

This consumer will read data from the beginning of the topic and write all the data to the console.

Console

Confluent control center

It's a browser-based UI. Open your browser and go to "http://localhost:9021". To start a consumer go to "Cluster 1 -> Topics" in the left panel and then select topic name which in our case is wikimedia. Go to the Messages tab and enter 0 for Offset. This will read data from the beginning of the topic.



Confluent control center

Kafka | Kafka Producer | Python | Data Engineering | Data Streaming

Written by Mrugank Ray

7 Followers

Follow

More from Mrugank Ray

Mrugank Ray

Real-time AVRO Data Analysis with Spark Streaming and Confluent...

Overview:

12 min read · Mar 25

25 1

Mrugank Ray

Unlocking the Vault of Tech Interview Questions

As a Senior Engineer who has gone through several job interviews in the tech industry an...

10 min read · Jun 18

16

Mrugank Ray

PostgreSQL vs MySQL | Choose What's Best For Your Project

Introduction :There are many options available when it comes to database system...

8 min read · Sep 25, 2022

1 1

See all from Mrugank Ray

Recommended from Medium

Ismael Sánchez Chaves in C# Programming

Schema Registry in Kafka: Avro, JSON and Protobuf

The importance of having a structured data schema for messaging-based systems

7 min read · May 15

24 1

Prateek

Avro Example 5 -use Date and Timestamp

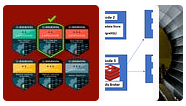
How to use Date and Timestamp in Avro, lets check that out —

3 min read · Jun 22

Lists

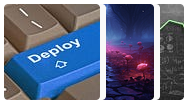
Coding & Development

11 stories · 258 saves



New_Reading_List

174 stories · 176 saves



Predictive Modeling w/ Python

20 stories · 561 saves

Practical Guides to Machine Learning

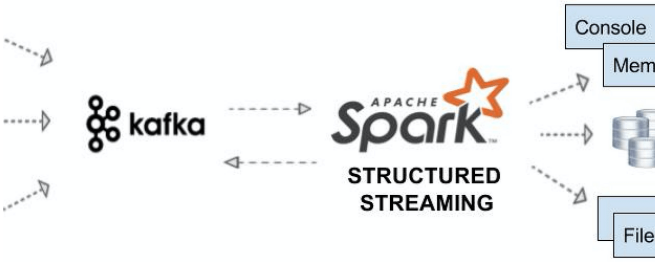
10 stories · 637 saves

Jay Kim

[Docker] Docker Compose example for Kafka, Zookeeper, and Schem...

1 Kafka Broker Setup

2 min read · Jun 1

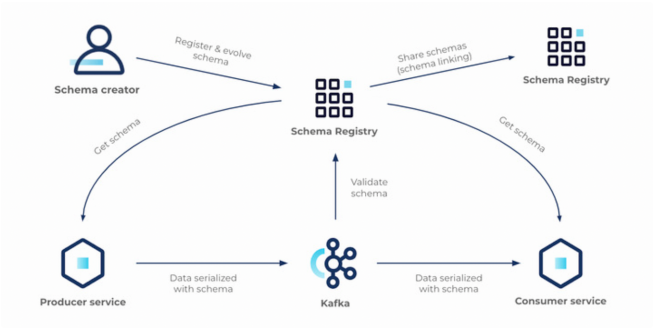


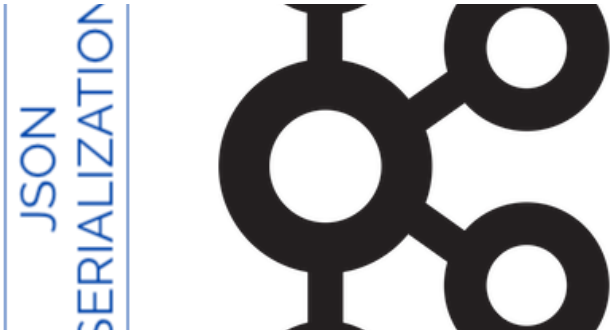
Ahmed Uz Zaman in Dev Genius

PySpark and Kafka Streaming: Reading and Writing Data in...

A Comprehensive Guide to Reading and Writing Data in Different Formats from Kafka...

· 4 min read · May 7





Kamini Kamal in Level Up Coding

Schema Registry in Kafka ecosystem

The Schema Registry is an integral component in the Kafka ecosystem, providin...

4 min read · Jul 2

Rob Golder in Lydtech Consulting

Kafka JSON Serialization

Consuming and producing messages in the JSON format is a popular choice for clients...

· 11 min read · Jun 17