Python Automation and Kafka Performance Metrics In Confluent Cloud

---

Sephora

Spring 2024

By Mason K. Yu Jr.

Executive Overview

This paper is to drive home and impress upon the readership the necessity and vitality to have robust Kafka client side analytics when dealing with the managed Kafka brokerage services of Confluent Cloud. The expressed readership and Kafka stakeholders must realize that Kafka has more than one management tool available for system reliability engineering. In most troubleshooting scenarios have several tools available can enable SRE's and Kafka support personnel valuable insight and buy time to ascertain the root cause of the

problem.

Recently there was a Jira posting addressed to me indicating a need for client side metrics for Confluent Cloud. By developing a robust troubleshooting diagnostic tool this time around can be leveraged into a broader base of tools for disaster recovery(DR drills as well). So this request is not just designed for this jira alone, but for future similar requests as well.

Control Center lacks the ability to act as a producer and consumer. Frankly, it is an albatross on lead wings which in some respects obfuscate the underlying problem at hand. Can it tell whether a topic has balanced partitions or not? What if we need to do a tail -50 and INSPECT what the key value pairs are ???

What about offset management and the integrity thereof pertaining to in-sync replicas (ISRs) stability ??? This paper is more than theory. It will deliberate upon the practicality of having a custom Kafka Swiss Army knife for a pittance.

This paper will outline the actual Python code with documentation, so that the Kafka team will have a formidable and solid basis moving forward.

The format is the type of metric and systemic functionality needed followed by the Python code itself. There will be also a brief explanation on how to use the tool.

The tool can evolve over time as Confluent makes upgrades to the Kafka ecosystem such as Flink.

Client side metrics fall under the broader category of SRE(System Reliability Engineering). In the next Section, a brief lineage of what the system requirements are articulated. Frankly, there are not a lot of moving parts required.

System requirements

As mentioned earlier in this paper, there are not a lot of moving parts. In the first place, there needs to be a lower environment to test out the Python automation – dev, qa or perf. Next, there needs to be a small Kafka Cluster built on Confluent Control Center. Basically, once the virtual Kafka cluster is built the key artifact needed going forward is the URL of the bootstrap servers of this virtual brokerage service.

Next, a small jump box will be required to develop, test and codify the Python scripts which is the "magic sauce" behind the automation and client side

metrics scraping. The Python code will import the mandatory Kafka libraries to communicate with the virtual broker service, use the URL link and that is all that is needed. The bulk of this paper is to demonstrate and document the client side metrics as necessary.

As part of the software stack, the OS preferably Ubuntu or RHE 7 or 8 will work. Pip will be mandatory as is a recent version of Python version 3.7 or so. The envisioned Python code code be manipulated and modified using vim or comparable text editor.

Index of Client Side Metric Tasks

Task4 Build out Python metric to check on the partitions whether they are Balanced within the topic mason using Kafka client side libraries

Task5 Find out how many messages were written out to a Kafka Topic called Mason and print out the most current message

Task6 Find out the high water mark(offsets) for each partition in a topic Called mason using Kafka client side libraries and print them out.

Task7 Sort in descending sequence the size of each topic in the Cluster of Kafka brokers

Task8 Sort in descending sequence the size of each topic in the

Kafka brokers Cluster. (please include replication factor as well)

Task9 Print out every 500th message from a Kafka topic called mason

Task10 How to check for consumer lag for a topic called mason and Whose Consumer-group is called mason-group.

Task11 How to check for leader/follower integrity and ISR count For a given topic

Task12 write out a timestamp & a monotonically increasing integer In the Kafka message header and print out to the Kafka console.

Task13 Search by integer in the message header from a given Kafka topic Called mason and print message on console.

Task14 Create two topics mason1 and mason2. Generate 50000 Messages per Topic{mason1 <-> mason2}

Task15 Print out the current message written to the topic called Mason and print the current message as well

Task16 Print out the current metadata for a topic called mason onto The console as well

Task17 Print out the current offset and number of messages by

Partition for A topic named mason onto the console.

Task18 Reset all the partitions for a topic named mason and reset The offset to the earliest, essentially resetting the topic partitions to The beginning

Task19 Time a Kafka consumer to consume 200000 messages From a topic called mason And get the Timings from start to Finish.

The Python Code for Client Side Metrics

(return to index of tasks)

Task1: Build out topic name mason, replication factor, number of partitions using Confluent Kafka libraries

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka.admin import AdminClient, NewTopic
```

```
def create_topic(bootstrap_servers, security_config, topic_name, num_partitions, replication_factor):
    # Merge the base configuration with the security configuration
    config = {
        'bootstrap.servers': bootstrap_servers,
        **security_config
    }

    admin_client = AdminClient(config)

    topic = NewTopic(
        topic_name,
        num_partitions=num_partitions,
        replication_factor=replication_factor  # Set based on your Confluent Cloud configuration
    )

    # Create the topic
    fs = admin_client.create_topics([topic])

    # Wait for each operation to finish.
    for topic, f in fs.items():
```

```
        try:
            f.result()  # The result itself is None
            print(f"Topic {topic} created")
        except Exception as e:
            print(f"Failed to create topic {topic}: {e}")

if __name__ == "__main__":
    bootstrap_servers = 'your_bootstrap_servers'  # Your Confluent Cloud cluster bootstrap servers
    security_config = {
        'sasl.mechanisms': 'PLAIN',
        'sasl.username': 'your_api_key',
        'sasl.password': 'your_api_secret',
        'security.protocol': 'SASL_SSL'
    }
    topic_name = 'mason'
    num_partitions = 6  # Desired number of partitions
    replication_factor = 3  # Desired replication factor (adjust based on your Confluent Cloud setup)

    create_topic(bootstrap_servers, security_config, topic_name, num_partitions, replication_factor)
```

(return to the index of tasks)

Task2: Build out Python Kafka Producer to write out 100000 messages To topic mason; also get the time it took to create these messages using Confluent Kafka libraries

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Producer
import time
```

```
def acked(err, msg):
    if err is not None:
        print(f"Failed to deliver message: {err}")
    else:
        print(f"Message produced: {msg.topic()}")

def produce_messages(producer, topic_name, count):
    for i in range(count):
        message = f"message {i}"
        producer.produce(topic_name, value=message, callback=acked)
        # Wait for all messages to be sent
        producer.poll(0)

    producer.flush()

if __name__ == "__main__":
    # Configuration for Kafka Producer
    config = {
        'bootstrap.servers': 'localhost:9092', # Update this to your Kafka server address
    }
```

```
    topic_name = 'your_topic_name' # Update this to your Kafka topic name
    message_count = 100000

    producer = Producer(**config)

    start_time = time.time()
    produce_messages(producer, topic_name, message_count)
    end_time = time.time()

    print(f"Produced {message_count} messages in {end_time - start_time} seconds.")
```

(return to the index of tasks)

Task3: Build out Python Kafka Producer to write out 100000 messages Asynchronously To topic mason; also get the time it took to create these messages
using Confluent Kafka libraries

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Producer
import json

def delivery_report(err, msg):
    """ Called once for each message produced to indicate delivery result.
        Triggered by poll() or flush(). """
    if err is not None:
        print('Message delivery failed: {}'.format(err))
    else:
        print('Message delivered to {} [{}]'.format(msg.topic(), msg.partition()))

def async_produce_messages(producer, topic_name, count):
    for i in range(count):
        message = f"async message {i}"
        # Trigger any available delivery report callbacks from previous produce() calls
        producer.poll(0)

        producer.produce(topic_name, message.encode('utf-8'), callback=delivery_report)

    # Wait for any outstanding messages to be delivered and delivery report
    # callbacks to be triggered.
    producer.flush()
```

```
if __name__ == "__main__":
    # Kafka configuration
    conf = {
        'bootstrap.servers': 'localhost:9092',  # Update this to your Kafka server address
        'client.id': 'python-producer'
    }

    topic_name = 'your_topic_name'  # Update this to your Kafka topic name
    message_count = 1000000

    producer = Producer(**conf)

    async_produce_messages(producer, topic_name, message_count)
```

(return to the index of tasks)

Task4: Build out Python metric to check on the partitions whether they are Balanced within the topic mason using Kafka client side libraries

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Producer, Consumer, KafkaException
```

```
def fetch_topic_metadata(broker, topic_name):
    # Configuration for Kafka client (Producer or Consumer can be used here for metadata fetching)
    conf = {
        'bootstrap.servers': broker,
        'client.id': 'partition-checker'
    }

    # Using a Consumer but not subscribing, just for fetching metadata
    client = Consumer(**conf)

    try:
        # Fetch metadata for the specified topic
        metadata = client.list_topics(topic=topic_name, timeout=10)
        topic_metadata = metadata.topics[topic_name]
        print(f"Metadata for topic '{topic_name}':")

        for partition_id, partition_info in topic_metadata.partitions.items():
            print(f"Partition: {partition_id}, Leader: {partition_info.leader}")
    except KafkaException as e:
        print(f"Failed to fetch metadata: {e}")
    finally:
```

```
        client.close()

if __name__ == "__main__":
    broker = 'localhost:9092'  # Update this to your Kafka broker address
    topic_name = 'your_topic_name'  # Update this to your Kafka topic name

    fetch_topic_metadata(broker, topic_name)
```

(return to the index of tasks)

Task5: Find out how many messages were written out to a Kafka topic called Mason and print out the most current message using Confluent Kafka libraries

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Consumer, KafkaError, OFFSET_END

def consume_latest_message(brokers, topic_name):
    # Consumer configuration
    # Note: Replace 'your_group_id' with an appropriate group ID
    conf = {
        'bootstrap.servers': brokers,
        'group.id': 'your_group_id',
        'auto.offset.reset': 'latest'
    }

    # Create Consumer instance
    consumer = Consumer(**conf)

    try:
        # Subscribe to the topic
        consumer.subscribe([topic_name], on_assign=on_assign)

        # Poll for a single message
        msg = consumer.poll(timeout=10.0)

        if msg is None:
```

```
            print("No new messages.")
        elif msg.error():
            if msg.error().code() == KafkaError._PARTITION_EOF:
                # End of partition event
                print('No more messages.')
            else:
                print(f"Error: {msg.error()}")
        else:
            # Message is a normal message
            print(f"Received message: {msg.value().decode('utf-8')}")
    finally:
        # Close down consumer to commit final offsets.
        consumer.close()

def on_assign(consumer, partitions):
    # Set offset to the end for each partition to consume the latest message only
    for partition in partitions:
        partition.offset = OFFSET_END
    consumer.assign(partitions)

if __name__ == "__main__":
    brokers = 'localhost:9092'  # Update this to your Kafka broker addresses
```

```
    topic_name = 'mason'  # The topic you want to consume from

    consume_latest_message(brokers, topic_name)
```

(return to the index of tasks)

Task6: Find out the high water mark(offsets) for each partition in a topic Called mason using Kafka client side libraries and print them out.

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Consumer, TopicPartition, KafkaException, AdminClient

def get_partition_offsets(brokers, topic_name):
    # Consumer configuration
    conf = {
        'bootstrap.servers': brokers,
        'group.id': 'partition_offsets_group',
        'auto.offset.reset': 'earliest'
    }

    # Create Consumer instance
    consumer = Consumer(**conf)

    # Create AdminClient instance
    admin_client = AdminClient({'bootstrap.servers': brokers})

    try:
        # Fetch metadata for the topic to get the partition information
        metadata = admin_client.list_topics(topic=topic_name, timeout=5)
        topic_metadata = metadata.topics[topic_name]
        partitions = topic_metadata.partitions
```

```
        # Prepare a list of TopicPartition objects with unset offsets
        topic_partitions = [TopicPartition(topic_name, p) for p in partitions]

        # Query for the high and low offsets for each partition
        low_high_offsets = consumer.get_watermark_offsets(topic_partitions, timeout=5,
cached=False)

        for partition, offsets in zip(topic_partitions, low_high_offsets):
            print(f"Partition: {partition.partition}, Low offset: {offsets[0]}, High offset (size):
{offsets[1]}")
    except KafkaException as e:
        print(f"An error occurred: {e}")
    finally:
        consumer.close()

if __name__ == "__main__":
    brokers = 'localhost:9092'  # Update this to your Kafka broker addresses
    topic_name = 'mason'  # The topic you're interested in

    get_partition_offsets(brokers, topic_name)
```

(return to the index of tasks)

Task7: Sort in descending sequence the size of each topic in the Kafka brokers Cluster.

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Consumer, TopicPartition, AdminClient

def get_all_topic_sizes(brokers):
    # Configuration for Kafka
    conf = {
        'bootstrap.servers': brokers,
        'group.id': 'size_check_group',
        'auto.offset.reset': 'earliest'
    }

    admin_client = AdminClient({'bootstrap.servers': brokers})
    consumer = Consumer(**conf)

    try:
        # Get metadata for all topics
        metadata = admin_client.list_topics(timeout=10)
        topics = metadata.topics

        topic_sizes = {}

        for topic in topics:
            # Skip internal topics
```

```
            if topic.startswith('_'):
                continue

            partitions = topics[topic].partitions
            total_size = 0

            # Fetch high offsets for each partition
            for p in partitions:
                topic_partition = [TopicPartition(topic, p, offset=0)]
                _, high_offset = consumer.get_watermark_offsets(topic_partition[0],
timeout=10, cached=False)
                total_size += high_offset

            topic_sizes[topic] = total_size

        # Sort topics by size in descending order
        sorted_topics = sorted(topic_sizes.items(), key=lambda x: x[1], reverse=True)

        for topic, size in sorted_topics:
            print(f"Topic: {topic}, Size (High Watermark Offset Sum): {size}")
    finally:
        consumer.close()
```

```
if __name__ == "__main__":
    brokers = 'localhost:9092'  # Update this to your Kafka broker addresses
    get_all_topic_sizes(brokers)
```

(return to the index of tasks)

Task8: Sort in descending sequence the size of each topic in the Kafka brokers Cluster. (please include replication factor as well)

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import AdminClient, Consumer, TopicPartition

def calculate_topic_sizes(brokers):
    admin_conf = {'bootstrap.servers': brokers}
    admin_client = AdminClient(admin_conf)

    consumer_conf = {
        'bootstrap.servers': brokers,
        'group.id': 'topic_size_calculation',
        'auto.offset.reset': 'earliest'
    }
    consumer = Consumer(consumer_conf)

    try:
        # Fetch metadata for all topics
        cluster_metadata = admin_client.list_topics(timeout=10)

        topic_sizes = {}

        for topic, topic_metadata in cluster_metadata.topics.items():
            if topic.startswith('_'):  # Ignore internal topics
                continue
```

```
            total_size = 0
            partitions = topic_metadata.partitions
            replication_factor = len(partitions[next(iter(partitions))].replicas)

            # Fetch high watermark offsets for each partition
            for partition_id in partitions:
                partition = TopicPartition(topic, partition_id)
                _, high_offset = consumer.get_watermark_offsets(partition, timeout=10,
cached=False)
                total_size += high_offset

            # Adjust total size by replication factor
            adjusted_size = total_size * replication_factor
            topic_sizes[topic] = adjusted_size

        # Sort topics by adjusted size in descending order
        sorted_topics = sorted(topic_sizes.items(), key=lambda item: item[1], reverse=True)

        for topic, size in sorted_topics:
            print(f"Topic: {topic}, Adjusted Size (Considering Replication Factor): {size}")
```

```
    finally:
        consumer.close()

if __name__ == "__main__":
    brokers = 'localhost:9092'  # Update with your Kafka broker address
    calculate_topic_sizes(brokers)
```

(return to the index of tasks)

Task9: Print out every 500th message from a Kafka topic called mason.

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Consumer, KafkaError
```

```
def consume_and_print_every_500th_message(brokers, topic_name):
    # Consumer configuration
    conf = {
        'bootstrap.servers': brokers,
        'group.id': 'group500',
        'auto.offset.reset': 'earliest'
    }

    # Create Consumer instance
    consumer = Consumer(**conf)
    consumer.subscribe([topic_name])

    try:
        message_count = 0
        while True:
            msg = consumer.poll(timeout=1.0)  # Adjust poll timeout as needed

            if msg is None:
                continue
            if msg.error():
                if msg.error().code() == KafkaError._PARTITION_EOF:
```

```
                # End of partition event
                print('End of partition reached')
            else:
                print(f"Error: {msg.error()}")
            continue

        message_count += 1
        if message_count % 500 == 0:
            print(f"Received message #{message_count}: {msg.value().decode('utf-8')}")

    except KeyboardInterrupt:
        print("Stopping consumer...")
    finally:
        # Clean up on exit
        consumer.close()

if __name__ == "__main__":
    brokers = 'localhost:9092'  # Update this to your Kafka broker addresses
    topic_name = 'mason'  # The topic you want to consume from

    consume_and_print_every_500th_message(brokers, topic_name)
```

(return to the index of tasks)

Task10: How to check for consumer lag for a topic called mason and whose Consumer-group is called mason-group.

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Consumer, TopicPartition, KafkaError, KafkaException
```

```
def check_consumer_lag(broker, topic_name, group_id):
    # Configuration for Kafka Consumer
    conf = {
        'bootstrap.servers': broker,
        'group.id': group_id,
        'auto.offset.reset': 'earliest',
        'enable.auto.commit': False,
    }

    consumer = Consumer(**conf)

    try:
        # Subscribe to the topic
        consumer.subscribe([topic_name])

        # Temporary assignment to get the partition information
        # This is a workaround to get partition info for a topic
        consumer.poll(timeout=1.0)
        partitions = consumer.assignment()
        consumer.unsubscribe()

        # Fetch the committed offsets for these partitions from the consumer group
```

```
        committed_offsets = consumer.committed(partitions, timeout=10)

        # Fetch the current end offsets (high watermark) for each partition
        end_offsets = consumer.get_watermark_offsets(partitions, timeout=10,
cached=False)

        # Calculate and print the lag for each partition
        for committed, (low, high) in zip(committed_offsets, end_offsets):
            partition_id = committed.partition
            committed_offset = committed.offset
            lag = high - committed_offset
            print(f"Partition: {partition_id}, Committed Offset: {committed_offset}, High
Watermark: {high}, Lag: {lag}")

    except KafkaException as e:
        print(f"An error occurred: {e}")
    finally:
        consumer.close()

if __name__ == "__main__":
    broker = 'localhost:9092'  # Update this to your Kafka broker address
    topic_name = 'mason'  # The topic you want to check
```

```
    group_id = 'mason-group'  # The consumer group ID
    check_consumer_lag(broker, topic_name, group_id)
```

(return to the index of tasks)

Task11: How to check for leader/follower integrity and ISR count

For a given topic

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import AdminClient, KafkaException

def check_leader_follower_integrity(broker, topic_name):
    admin_client = AdminClient({'bootstrap.servers': broker})

    try:
        # Fetch metadata for the specified topic
        metadata = admin_client.list_topics(topic=topic_name, timeout=10)
        topic_metadata = metadata.topics[topic_name]
        print(f"Checking leader-follower integrity for topic '{topic_name}'")

        for partition_id, partition_info in topic_metadata.partitions.items():
            leader = partition_info.leader
            replicas = partition_info.replicas
            isr = partition_info.isr

            print(f"\nPartition: {partition_id}")
            print(f"Leader: {leader}")
            print(f"Replicas: {replicas}")
            print(f"In-Sync Replicas (ISR): {isr}")
```

```
            # Checking if the leader is in the list of in-sync replicas
            if leader not in isr:
                print("Warning: Leader is not in the ISR list, which could indicate a problem.")
            else:
                print("Leader is in the ISR list.")

            # Optionally, check if all replicas are in the ISR
            if set(replicas).issubset(set(isr)):
                print("All replicas are in the ISR list.")
            else:
                print("Warning: Not all replicas are in the ISR list, which could indicate a
replication problem.")

    except KafkaException as e:
        print(f"An error occurred: {e}")

if __name__ == "__main__":
    broker = 'localhost:9092'  # Update this to your Kafka broker address
    topic_name = 'mason'  # The topic you want to check

    check_leader_follower_integrity(broker, topic_name)
```

(return to the index of tasks)

Task12: write out a timestamp and a monotonically increasing integer in the Kafka message header and print out to the Kafka console.

Pip: `pip install confluent_kafka`

Python Code:

```
from datetime import datetime, timedelta
from confluent_kafka import Producer
import json

def delivery_report(err, msg):
    """Called once for each message produced to indicate delivery result."""
    if err is not None:
        print(f"Message delivery failed: {err}")
    else:
        print(f"Message delivered to {msg.topic()} [{msg.partition()}]")

def generate_and_send_messages(producer, topic_name, count=10000):
    start_date = datetime.now()

    for i in range(count):
        message_id = i + 1  # Monotonically increasing positive integer
        timestamp = (start_date + timedelta(milliseconds=i)).strftime("%Y-%m-%d
%H:%M:%S.%f")
```

```
        message = {
            "header": {
                "timestamp": timestamp,
                "message_id": message_id
            },
            "body": "This is message body"
        }

        # Print message to console
        print(message)

        # Convert message to a string format before sending
        message_str = json.dumps(message)

        # Send message to Kafka
        producer.produce(topic_name, message_str.encode('utf-8'),
callback=delivery_report)

        # Wait up to 1 second for events. Callbacks will be invoked during
        # this method call if the message is acknowledged.
        producer.poll(1)
```

```
if __name__ == "__main__":
    brokers = 'localhost:9092'  # Update this to your Kafka broker addresses
    topic_name = 'mason'

    # Kafka producer configuration
    conf = {
        'bootstrap.servers': brokers
    }

    producer = Producer(**conf)

    generate_and_send_messages(producer, topic_name)

    # Wait for all messages to be delivered
    producer.flush()

    print("All messages have been sent to Kafka")
```

(return to the index of tasks)

Task13: Search by integer in the message header from a given Kafka topic Called mason and print message on console.

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Consumer, KafkaError

def find_message_by_header_integer(broker, topic_name, group_id, search_integer):
    # Consumer configuration
    conf = {
        'bootstrap.servers': broker,
        'group.id': group_id,
        'auto.offset.reset': 'earliest'
    }

    consumer = Consumer(**conf)
    consumer.subscribe([topic_name])

    try:
        while True:
            msg = consumer.poll(1.0)  # Adjust poll timeout as needed

            if msg is None:
                continue
            if msg.error():
                if msg.error().code() == KafkaError._PARTITION_EOF:
                    # End of partition event
```

```
                print('End of partition reached.')
            else:
                print(f"Error: {msg.error()}")
            continue

        # Check if any header matches the search criteria
        headers = msg.headers()
        for key, value in headers:
            if key == "message_id":
                message_id = int(value.decode('utf-8'))  # Assuming the header value is
encoded as a string
                if message_id == search_integer:
                    print(f"Found message with ID {search_integer}:
{msg.value().decode('utf-8')}")
                    return  # Exit after finding the message

    except KeyboardInterrupt:
        print("Search interrupted by user.")
    finally:
        consumer.close()

if __name__ == "__main__":
```

```
    broker = 'localhost:9092'  # Your Kafka broker address
    topic_name = 'mason'  # The topic to search
    group_id = 'search-group'  # Consumer group ID
    search_integer = 123  # The integer to search for in the message header

    find_message_by_header_integer(broker, topic_name, group_id, search_integer)
```

(return to the index of tasks)

Task14: Create two topics mason1 and mason2. Generate 50000 message per Topic{mason1 <- mason2}

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Producer
import time

def acked(err, msg):
    if err is not None:
        print(f"Failed to deliver message: {err}")
    else:
        print(f"Message delivered to {msg.topic()} [{msg.partition()}]")

def produce_messages(producer, topic_names, count):
    for i in range(count):
        message = f"message {i}"
        for topic_name in topic_names:
            producer.produce(topic_name, value=message, callback=acked)
            # Asynchronously produce a message, the delivery report callback
            # will be triggered from poll() below, when the message has
            # been successfully delivered or failed permanently.
        producer.poll(0)  # Serve delivery callback queue.

    producer.flush()  # Wait for all messages to be delivered.

if __name__ == "__main__":
```

```
    # Configuration for Kafka Producer
    config = {
        'bootstrap.servers': 'localhost:9092',  # Update this to your Kafka server address
    }

    topic_names = ['mason1', 'mason2']  # Topics to produce messages to

    producer = Producer(**config)

    start_time = time.time()
    produce_messages(producer, topic_names, 50000)
    end_time = time.time()

    print(f"Produced 50,000 messages to each topic in {end_time - start_time:.2f}
seconds.")
```

(return to the index of tasks)

Task15: Print out the current message written to the topic called mason and print out its offset as well.

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Consumer, OFFSET_END

def consume_latest_message(broker, topic_name, group_id):
    # Consumer configuration
    conf = {
        'bootstrap.servers': broker,
        'group.id': group_id,
        'auto.offset.reset': 'earliest'
    }

    consumer = Consumer(**conf)
    consumer.subscribe([topic_name], on_assign=on_assign)

    try:
        # Poll for a limited time only as we're looking for the latest message
        # and don't want to start a long-running consumption
        msg = consumer.poll(5.0)  # Adjust timeout as needed

        if msg is None:
            print("No new messages.")
        elif msg.error():
            print(f"Error: {msg.error()}")
```

```
        else:
            # Message is a normal message
            print(f"Received message: {msg.value().decode('utf-8')}, Offset: {msg.offset()},
Partition: {msg.partition()}")

    finally:
        # Clean up on exit
        consumer.close()

def on_assign(consumer, partitions):
    # Adjusting the offset to the end for each partition to consume the latest message only
    for p in partitions:
        p.offset = OFFSET_END
    consumer.assign(partitions)

if __name__ == "__main__":
    broker = 'localhost:9092'  # Your Kafka broker address
    topic_name = 'mason'  # The topic you want to consume from
    group_id = 'latest-message-group'  # A unique group ID for this consumer

    consume_latest_message(broker, topic_name, group_id)
```

(return to the index of tasks)

Task16: Print out the current metadata for a topic called mason onto the console.

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Producer
```

```
def print_topic_metadata(broker, topic_name):
    # Configuration for Kafka Producer (can also use Consumer for this purpose)
    conf = {
        'bootstrap.servers': broker,
    }

    # Create a Producer (or Consumer) just to get the metadata
    producer = Producer(**conf)

    # Fetch metadata for the specific topic
    metadata = producer.list_topics(topic=topic_name, timeout=5)
    topic_metadata = metadata.topics[topic_name]

    print(f"Metadata for topic '{topic_name}':")

    # Print details for each partition in the topic
    for partition_id, partition_metadata in topic_metadata.partitions.items():
        print(f"\nPartition: {partition_id}")
        print(f"Leader: {partition_metadata.leader}")
        print(f"Replicas: {partition_metadata.replicas}")
        print(f"ISRs: {partition_metadata.isrs}")
```

```
    # Close the producer
    producer.flush()

if __name__ == "__main__":
    broker = 'localhost:9092'  # Update this to your Kafka server address
    topic_name = 'mason'  # Topic for which to fetch metadata

    print_topic_metadata(broker, topic_name)
```

(return to the index of tasks)

Task17: Print out the current offset and number of messages by partition for A topic named mason onto the console.

Pip: `pip install confluent_kafka`

Python Code:

```
from kafka import KafkaConsumer, TopicPartition

# Replace 'localhost:9092' with the address of your Kafka broker
kafka_broker_address = 'localhost:9092'
```

```
# Create a Kafka consumer without subscribing to any topic
consumer = KafkaConsumer(bootstrap_servers=[kafka_broker_address])

# Get the partitions for the topic 'mason'
partitions = consumer.partitions_for_topic('mason')

if partitions is not None:
    for partition in partitions:
        # Create a TopicPartition object for each partition
        tp = TopicPartition('mason', partition)

        # Get the last offset for each partition
        consumer.assign([tp])
        consumer.seek_to_end(tp)
        last_offset = consumer.position(tp)

        # The last offset gives us the number of messages in the partition, as offset starts at 0
        print(f'Partition {partition}: Current offset is {last_offset}, Number of messages is
{last_offset}')
else:
    print('Topic "mason" does not exist or has no partitions.')
```

```
# Always remember to close the consumer when done
consumer.close()
```

(return to the index of tasks)

Task18: Reset the partitions for a topic named mason and reset the offset to the Earliest, essentially resetting the topic partitions to the beginning

Pip: `pip install confluent_kafka`

Python Code:

```
from kafka import KafkaConsumer, TopicPartition

# Configuration
kafka_broker_address = 'localhost:9092'
topic_name = 'mason'
consumer_group = 'your_consumer_group'
```

```
# Initialize a consumer
consumer = KafkaConsumer(
    bootstrap_servers=[kafka_broker_address],
    group_id=consumer_group,
    auto_offset_reset='earliest',  # Automatically reset offset to earliest if it is out of range
    enable_auto_commit=False,  # Disable auto-commit. We'll manually commit the
offsets
)

# Subscribe to the topic
consumer.subscribe([topic_name])

# Get the partitions for the topic
partitions = consumer.partitions_for_topic(topic_name)
if partitions is not None:
    topic_partitions = [TopicPartition(topic_name, p) for p in partitions]

    # Seek to the earliest offsets for each partition
    for tp in topic_partitions:
        consumer.assign([tp])
        consumer.seek_to_beginning(tp)
```

```
    # Assuming you might want to do something here, like re-processing old messages
    # ...

    # Commit the offsets so that this new position is remembered by the consumer group
    consumer.commit()

    print(f'Reset offsets for topic "{topic_name}" to the earliest position for consumer group "{consumer_group}".')
else:
    print(f'Topic "{topic_name}" does not exist or has no partitions.')

# Close the consumer
consumer.close()
```

(return to the index of tasks)

Task19: Time a Kafka consumer to consume 200000 messages and get the Timings from start to finish.

Pip: `pip install confluent_kafka`

Python Code:

```
from confluent_kafka import Consumer, KafkaError
import time

# Kafka consumer configuration
config = {
    'bootstrap.servers': 'your_kafka_broker',  # Change this to your broker's address
    'group.id': 'your_consumer_group',         # Change this to your consumer group
    'auto.offset.reset': 'earliest'            # Start reading at the earliest message
}

# Initialize the consumer
consumer = Consumer(**config)
consumer.subscribe(['mason'])

# Variable to count messages
message_count = 0
```

```
# Record the start time
start_time = time.time()

try:
    while True:
        # Try to consume a message
        msg = consumer.poll(timeout=1.0)  # Adjust timeout as needed

        # Check for end of partition
        if msg is None:
            continue

        # Check for errors
        if msg.error():
            if msg.error().code() == KafkaError._PARTITION_EOF:
                # End of partition event
                continue
            else:
                # Actual error
                print(msg.error())
                break
```

```
        # Increment message count
        message_count += 1

        # Check if we've reached 200,000 messages
        if message_count >= 200000:
            break
finally:
    # Always close the consumer cleanly
    consumer.close()

# Record the end time
end_time = time.time()

# Calculate and print the duration
duration = end_time - start_time
print(f"Read 200,000 messages in {duration} seconds.")
```