Kruti Patel (202318032)

Delivery Processing System Report

To stay ahead in today's cutthroat market, businesses need lightning-fast delivery processing. Apache Kafka, a powerful distributed streaming platform, offers the key to unlocking this efficiency. This report details how we implemented Kafka producers and consumers, coupled with intelligent message filtering, to create a seamless system for managing inventory and delivery orders.

1.Kafka Producers Implementation:

Producer 1: Inventory Orders Producer:

Leveraging Kafka client libraries, the application establishes a connection to the Kafka broker and selectively publishes messages to the "inventory_orders" topic. A filtering mechanism ensures only messages categorized as "inventory" are transmitted, streamlining data flow and optimizing message delivery.

```python
from confluent_kafka import Producer
import json

file_path = '/FileStore/shared_uploads/202318032@daiict.ac.in/data-2.json'
df = spark.read.json(file_path)

# Define Kafka producer configuration
bootstrap_servers = 'localhost:9092'
producer_config = {
    'bootstrap.servers': bootstrap_servers
}
inventory_producer = Producer(producer_config)

# Function to produce a message for inventory orders
def produce_inventory_order(order):
    message = json.dumps(order)
    inventory_producer.produce('inventory_orders', message.encode('utf-8'))

# Assuming `df` is the DataFrame containing orders
for row in df.collect():
    order = row.asDict()
    if order['type'] == 'inventory':
        produce_inventory_order(order)

# Flush producer to ensure all messages are sent
inventory_producer.flush()
```

Producer 2: Delivery Orders Producer

Created a Kafka producer responsible for handling messages related to the delivery of orders.

Incorporated filtering functionality to send only messages where the type is "delivery".

Integrated error handling mechanisms to ensure reliable message delivery.

```
# Create Kafka producer for delivery orders
delivery_producer = Producer(producer_config)

# Function to produce a message for delivery orders
def produce_delivery_order(order):
    message = json.dumps(order)
    delivery_producer.produce('delivery_orders', message.encode('utf-8'))

for row in df.collect():
    order = row.asDict()
    if order['type'] == 'delivery':
        produce_delivery_order(order)

# Flush producer to ensure all messages are sent
delivery_producer.flush()
```

2. Kafka Consumers Implementation:

Consumer 1: Inventory Data Consumer:

Developed a Kafka consumer application to handle inventory data.

Configured the consumer to listen for messages with the type "inventory".

Implemented logic to process messages and update inventory databases or systems accordingly.

```python
from confluent_kafka import Consumer, KafkaError
consumer_config = {
    'bootstrap.servers': bootstrap_servers,
    'group.id': 'inventory_consumer_group',
    'auto.offset.reset': 'earliest'
}

# Create Kafka consumer for inventory data
inventory_consumer = Consumer(consumer_config)
inventory_consumer.subscribe(['inventory_orders'])

# Function to handle inventory data
def handle_inventory_data(order):
    # Process inventory data here
    print("Processing inventory data:", order)

# Poll for new messages in the Kafka topic and process them
try:
    while True:
        msg = inventory_consumer.poll(timeout=1.0)
        if msg is None:
            continue
        if msg.error():
            if msg.error().code() == KafkaError._PARTITION_EOF:
                continue
            else:
                print(msg.error())
                break
        order = json.loads(msg.value().decode('utf-8'))
        if order['type'] == 'inventory':
            handle_inventory_data(order)
```

```python

except KeyboardInterrupt:
    pass

finally:
    # Close Kafka consumer
    inventory_consumer.close()
```

Consumer 2: Delivery Data Consumer:

Implemented a Kafka consumer to manage delivery tasks.

Configured the consumer to listen for messages with the type "delivery".

Developed functionalities to schedule deliveries, update delivery status, and notify customers based on received messages.

```python
delivery_consumer = Consumer(consumer_config)
delivery_consumer.subscribe(['delivery_orders'])

def handle_delivery_data(order):
    print("Processing delivery data:", order)

try:
    while True:
        msg = delivery_consumer.poll(timeout=1.0)
        if msg is None:
            continue
        if msg.error():
            if msg.error().code() == KafkaError._PARTITION_EOF:
                continue
            else:
                print(msg.error())
                break
        order = json.loads(msg.value().decode('utf-8'))
        if order['type'] == 'delivery':
            handle_delivery_data(order)

except KeyboardInterrupt:
    pass

finally:
    delivery_consumer.close()
```