

Name: Tanaz Pathan
Student ID: 202318056

Real-Time E-commerce Order Processing System Using Kafka

Introduction:

This report outlines the development of a Kafka-based system for managing e-commerce orders in real time. The system handles inventory management and delivery processing by implementing Kafka producers and consumers.

Kafka Installation:

The setup involved installing the necessary libraries and configuring Kafka producers to send messages to Kafka topics.

- Installed `kafka-python` and `confluent-kafka` libraries using `pip`.
- Configured Kafka producers for inventory orders and delivery orders.
- Defined a function to acknowledge message delivery and flush messages to ensure they are sent.

Producer Implementation:

Two Kafka producers were implemented to send messages for inventory orders and delivery orders. Each producer sends messages with a specific type to the respective Kafka topics.

- Implemented a producer for inventory orders to send messages to the `'inventory_topic'`.
- Implemented a producer for delivery orders to send messages to the `'delivery_topic'`.



+ Code + Text

```
[3] conf = {
    'bootstrap.servers': "localhost:9092",
    'client.id': socket.gethostname()
}

# Function to acknowledge message delivery
def acked(err, msg):
    if err is not None:
        print("Failed to deliver message: %s: %s" % (str(msg), str(err)))
    else:
        print("Message produced: %s" % (str(msg)))

# Inventory Orders Producer
producer_inventory = Producer(conf)
def send_inventory_message(data):
    producer_inventory.produce('inventory_topic', key=str(data['order_id']), value=str(data), callback=acked)
    producer_inventory.poll(0)

# Delivery Orders Producer
producer_delivery = Producer(conf)
def send_delivery_message(data):
    producer_delivery.produce('delivery_topic', key=str(data['order_id']), value=str(data), callback=acked)
    producer_delivery.poll(0)

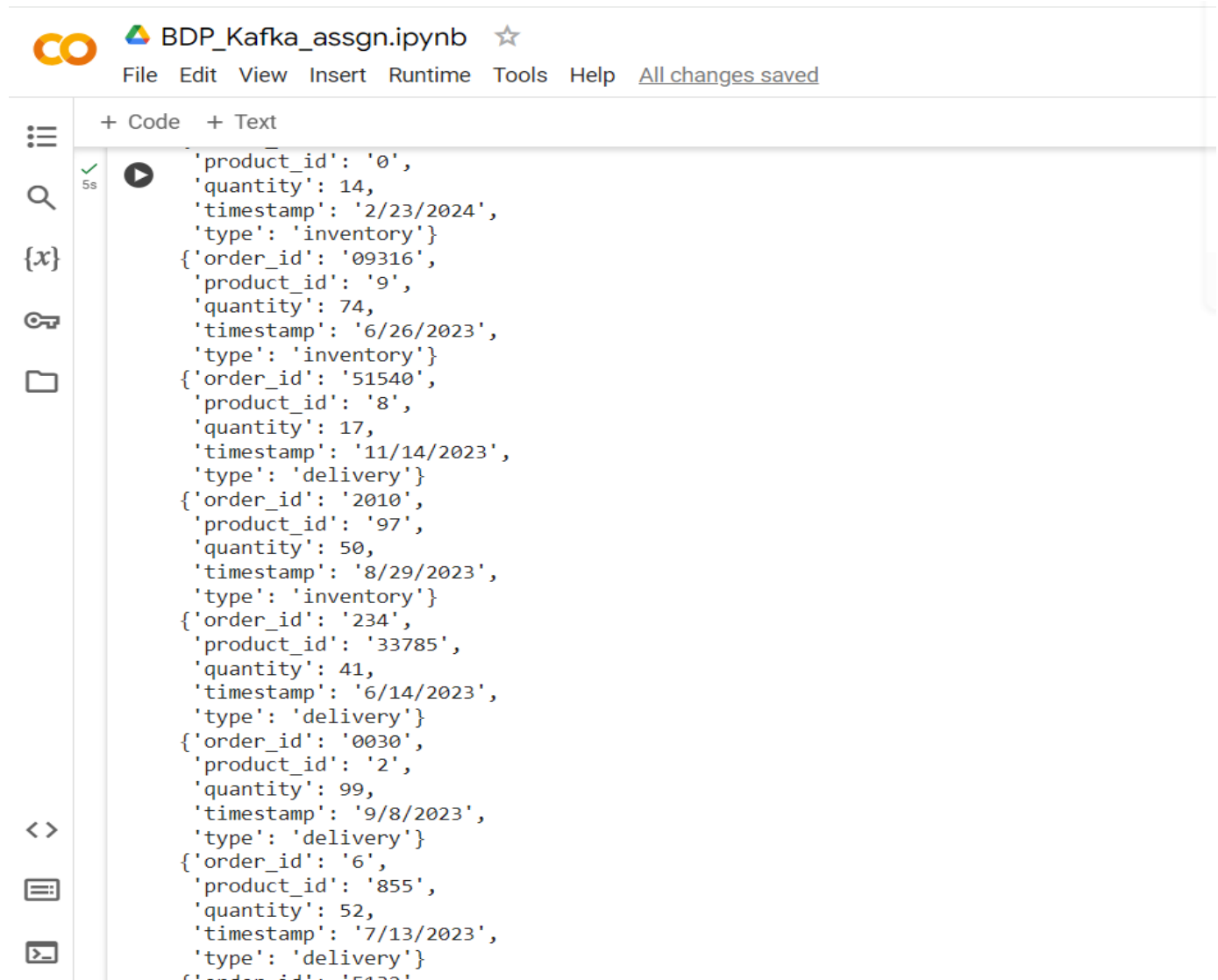
# Ensure all messages are sent
producer_inventory.flush()
producer_delivery.flush()
```

0

Message Sending:

Messages were sent to Kafka topics using the implemented producers. Each message contains order details such as order ID, product ID, quantity, and timestamp.

- Inventory messages were sent to the 'inventory_topic'.
- Delivery messages were sent to the 'delivery_topic'.



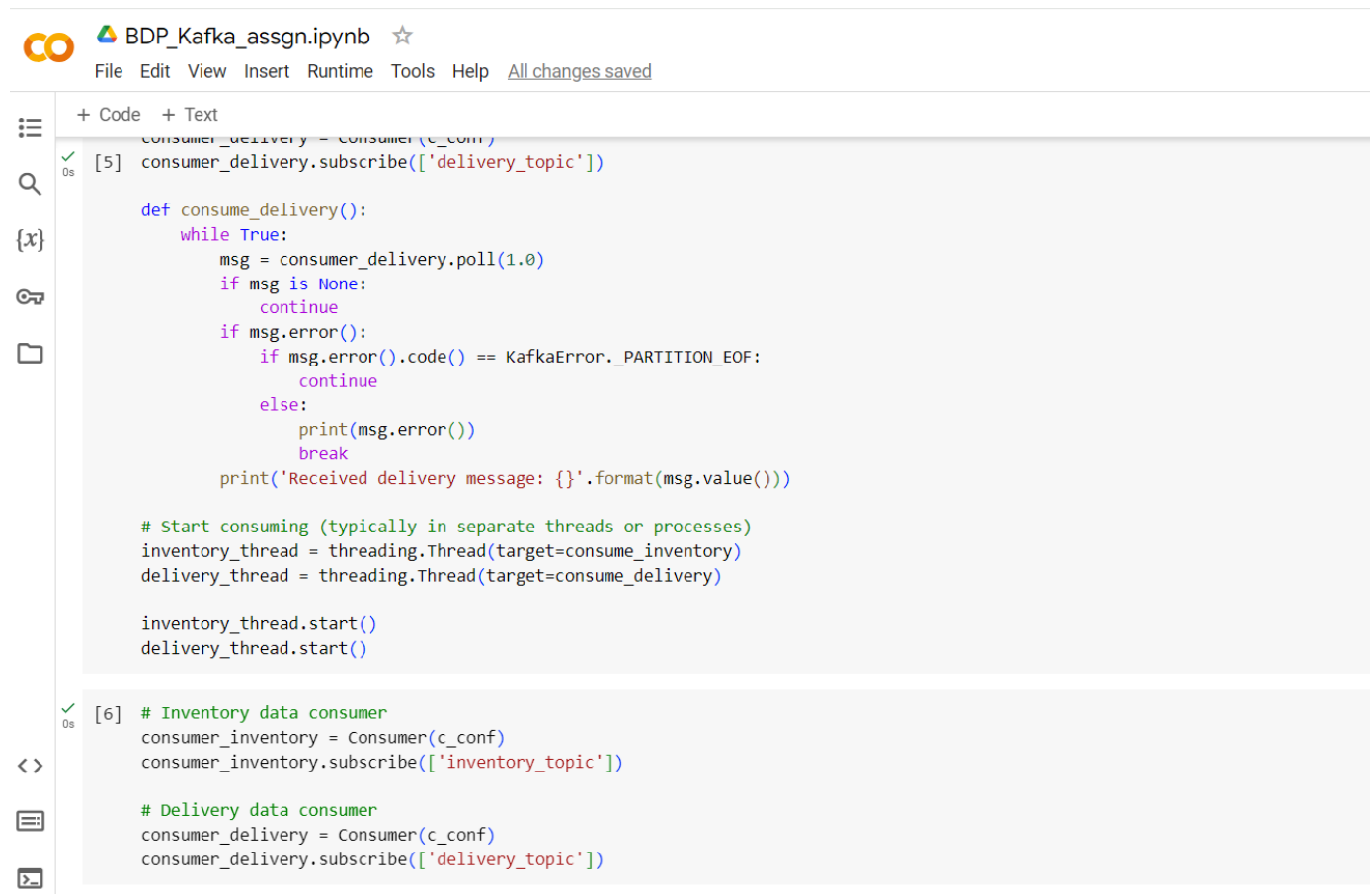
The screenshot shows a Jupyter Notebook titled "BDP_Kafka_assgn.ipynb". The interface includes a top bar with the file name and a star icon, and a menu bar with options: File, Edit, View, Insert, Runtime, Tools, Help, and a status indicator "All changes saved". On the left side, there is a sidebar with icons for a menu, search, variables, keys, and files. The main area displays a code cell with a play button icon and a "5s" execution time. The code cell contains a list of JSON messages, each representing an order. The messages are as follows:

```
'product_id': '0',  
'quantity': 14,  
'timestamp': '2/23/2024',  
'type': 'inventory'}  
{  
  'order_id': '09316',  
  'product_id': '9',  
  'quantity': 74,  
  'timestamp': '6/26/2023',  
  'type': 'inventory'  
}  
{  
  'order_id': '51540',  
  'product_id': '8',  
  'quantity': 17,  
  'timestamp': '11/14/2023',  
  'type': 'delivery'  
}  
{  
  'order_id': '2010',  
  'product_id': '97',  
  'quantity': 50,  
  'timestamp': '8/29/2023',  
  'type': 'inventory'  
}  
{  
  'order_id': '234',  
  'product_id': '33785',  
  'quantity': 41,  
  'timestamp': '6/14/2023',  
  'type': 'delivery'  
}  
{  
  'order_id': '0030',  
  'product_id': '2',  
  'quantity': 99,  
  'timestamp': '9/8/2023',  
  'type': 'delivery'  
}  
{  
  'order_id': '6',  
  'product_id': '855',  
  'quantity': 52,  
  'timestamp': '7/13/2023',  
  'type': 'delivery'  
}  
{  
  'order_id': '5122',  
  'product_id': '8',  
  'quantity': 17,  
  'timestamp': '11/14/2023',  
  'type': 'delivery'
```

Consumer Implementation:

Consumers were implemented to consume messages from Kafka topics for inventory data and delivery data. Each consumer listens to its respective Kafka topic and processes incoming messages.

- Implemented consumers for inventory data and delivery data.
- Subscribed consumers to the 'inventory_topic' and 'delivery_topic', respectively.
- Defined functions to consume and process incoming messages.



```
BDP_Kafka_assgn.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[5] consumer_delivery = Consumer(c_conf)
    consumer_delivery.subscribe(['delivery_topic'])

    def consume_delivery():
        while True:
            msg = consumer_delivery.poll(1.0)
            if msg is None:
                continue
            if msg.error():
                if msg.error().code() == KafkaError._PARTITION_EOF:
                    continue
                else:
                    print(msg.error())
                    break
            print('Received delivery message: {}'.format(msg.value()))

    # Start consuming (typically in separate threads or processes)
    inventory_thread = threading.Thread(target=consume_inventory)
    delivery_thread = threading.Thread(target=consume_delivery)

    inventory_thread.start()
    delivery_thread.start()

[6] # Inventory data consumer
    consumer_inventory = Consumer(c_conf)
    consumer_inventory.subscribe(['inventory_topic'])

    # Delivery data consumer
    consumer_delivery = Consumer(c_conf)
    consumer_delivery.subscribe(['delivery_topic'])
```

Message Processing:

Incoming messages were processed by the consumers to update inventory databases, schedule deliveries, update delivery status, and notify customers as per the message type.

- Inventory data consumers processed messages from the 'inventory_topic'.
- Delivery data consumers processed messages from the 'delivery_topic'.

 BDP_Kafka_assgn.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

+

 Code

+

 Text

✓ 0s

[5]

```
msg = consumer_inventory.poll(1.0)
if msg is None:
    continue
if msg.error():
    if msg.error().code() == KafkaError._PARTITION_EOF:
        continue
    else:
        print(msg.error())
        break
print('Received inventory message: {}'.format(msg.value()))

# Delivery Data Consumer
consumer_delivery = Consumer(c_conf)
consumer_delivery.subscribe(['delivery_topic'])

def consume_delivery():
    while True:
        msg = consumer_delivery.poll(1.0)
        if msg is None:
            continue
        if msg.error():
            if msg.error().code() == KafkaError._PARTITION_EOF:
                continue
            else:
                print(msg.error())
                break
        print('Received delivery message: {}'.format(msg.value()))

# Start consuming (typically in separate threads or processes)
inventory_thread = threading.Thread(target=consume_inventory)
delivery_thread = threading.Thread(target=consume_delivery)
```

✓ 0s completed at 10:53 PM

Conclusion:

The real-time e-commerce order processing system using Kafka was successfully implemented. Kafka producers and consumers were configured to handle inventory and delivery orders, ensuring efficient management of e-commerce operations.