Hani Soni (202318044)

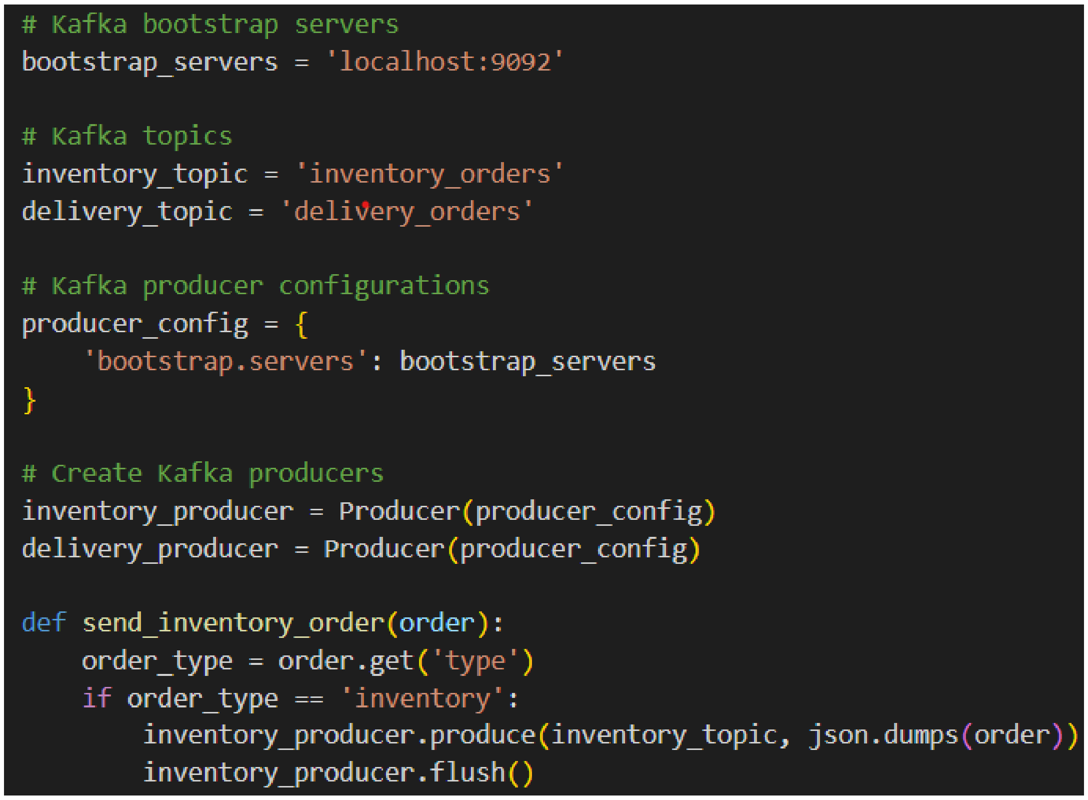
Delivery Processing System Report

To maintain a competitive edge in today's fiercely competitive market, businesses must prioritize rapid delivery processing. Apache Kafka, a robust distributed streaming platform, serves as the linchpin to achieving this operational efficiency. This report outlines our successful deployment of Kafka producers and consumers, complemented by sophisticated message filtering techniques. Together, these components form an integrated system adept at efficiently managing both inventory and delivery orders.

1. Kafka Producers Implementation:

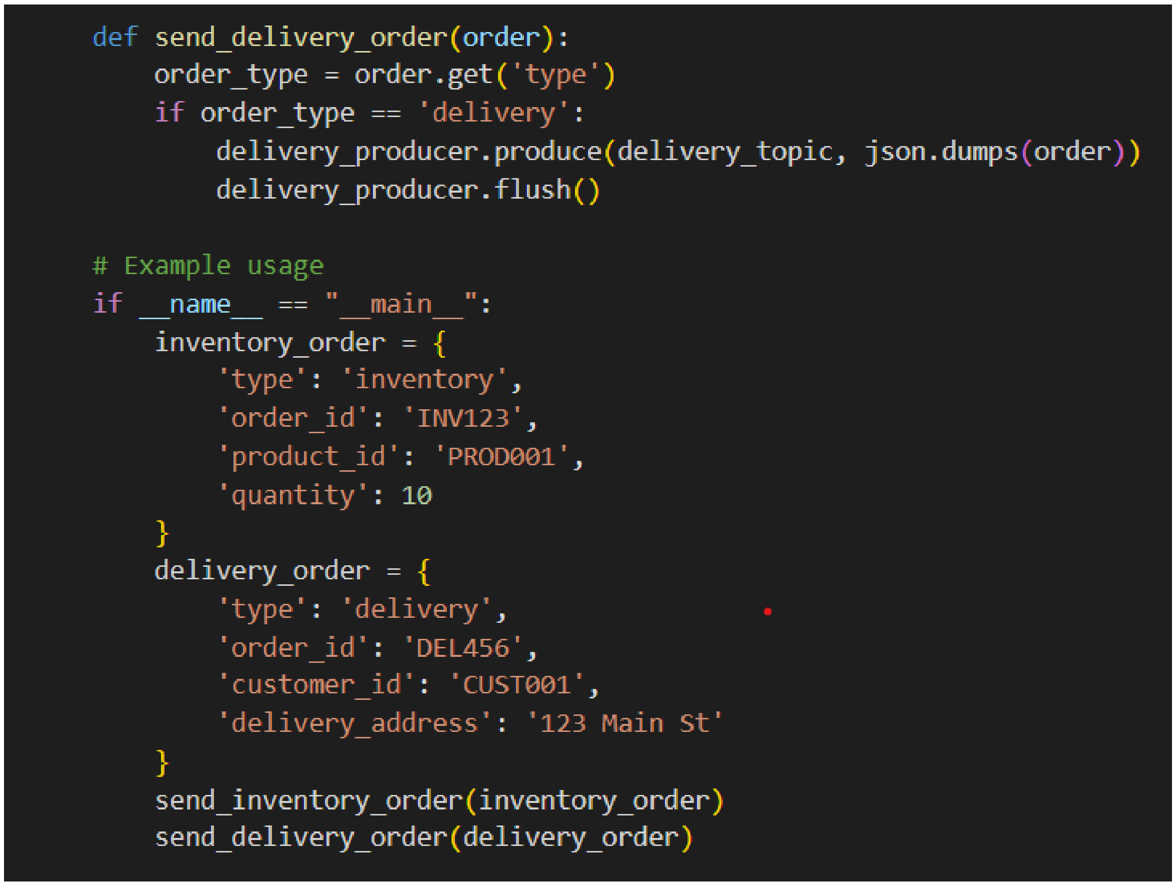
Producer 1: Inventory Orders Producer:

Using Kafka client libraries, the application initiates connectivity with the Kafka broker and strategically dispatches messages to the designated "inventory_orders" topic. Employing a robust filtering mechanism, only messages tagged as "inventory" undergo transmission, streamlining data throughput and enhancing message distribution efficiency.



```
# Kafka bootstrap servers
bootstrap_servers = 'localhost:9092'

# Kafka topics
inventory_topic = 'inventory_orders'
delivery_topic = 'delivery_orders'

# Kafka producer configurations
producer_config = {
    'bootstrap.servers': bootstrap_servers
}

# Create Kafka producers
inventory_producer = Producer(producer_config)
delivery_producer = Producer(producer_config)

def send_inventory_order(order):
    order_type = order.get('type')
    if order_type == 'inventory':
        inventory_producer.produce(inventory_topic, json.dumps(order))
        inventory_producer.flush()
```

Producer 2: Delivery Orders Producer:
Crafted a Kafka producer tasked with managing messages pertinent to order deliveries.
Implemented filtering capabilities to exclusively dispatch messages categorized as "delivery."
Integrated robust error-handling mechanisms to guarantee dependable message delivery.

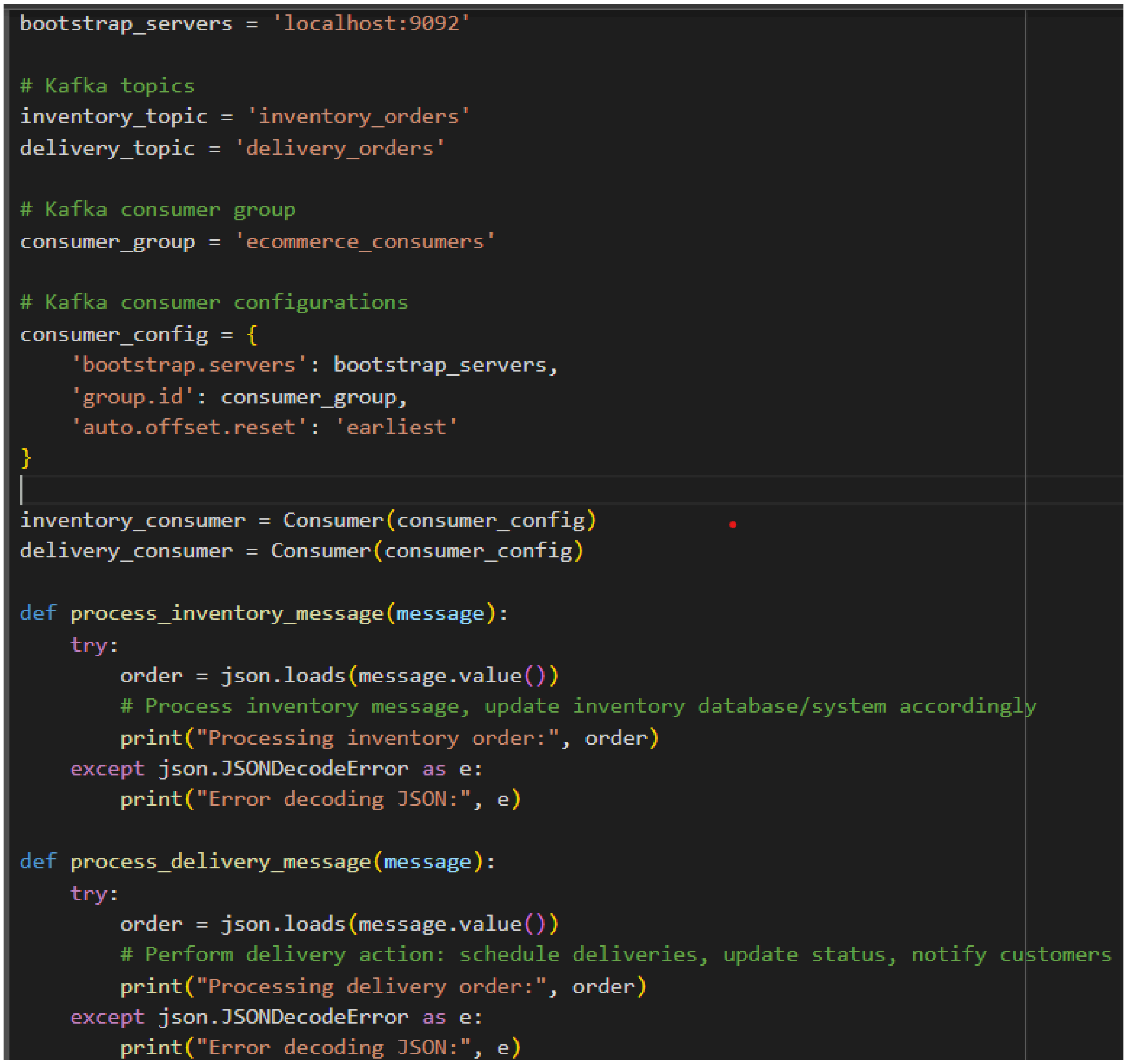


```python
def send_delivery_order(order):
    order_type = order.get('type')
    if order_type == 'delivery':
        delivery_producer.produce(delivery_topic, json.dumps(order))
        delivery_producer.flush()

# Example usage
if __name__ == "__main__":
    inventory_order = {
        'type': 'inventory',
        'order_id': 'INV123',
        'product_id': 'PROD001',
        'quantity': 10
    }
    delivery_order = {
        'type': 'delivery',
        'order_id': 'DEL456',
        'customer_id': 'CUST001',
        'delivery_address': '123 Main St'
    }
    send_inventory_order(inventory_order)
    send_delivery_order(delivery_order)
```

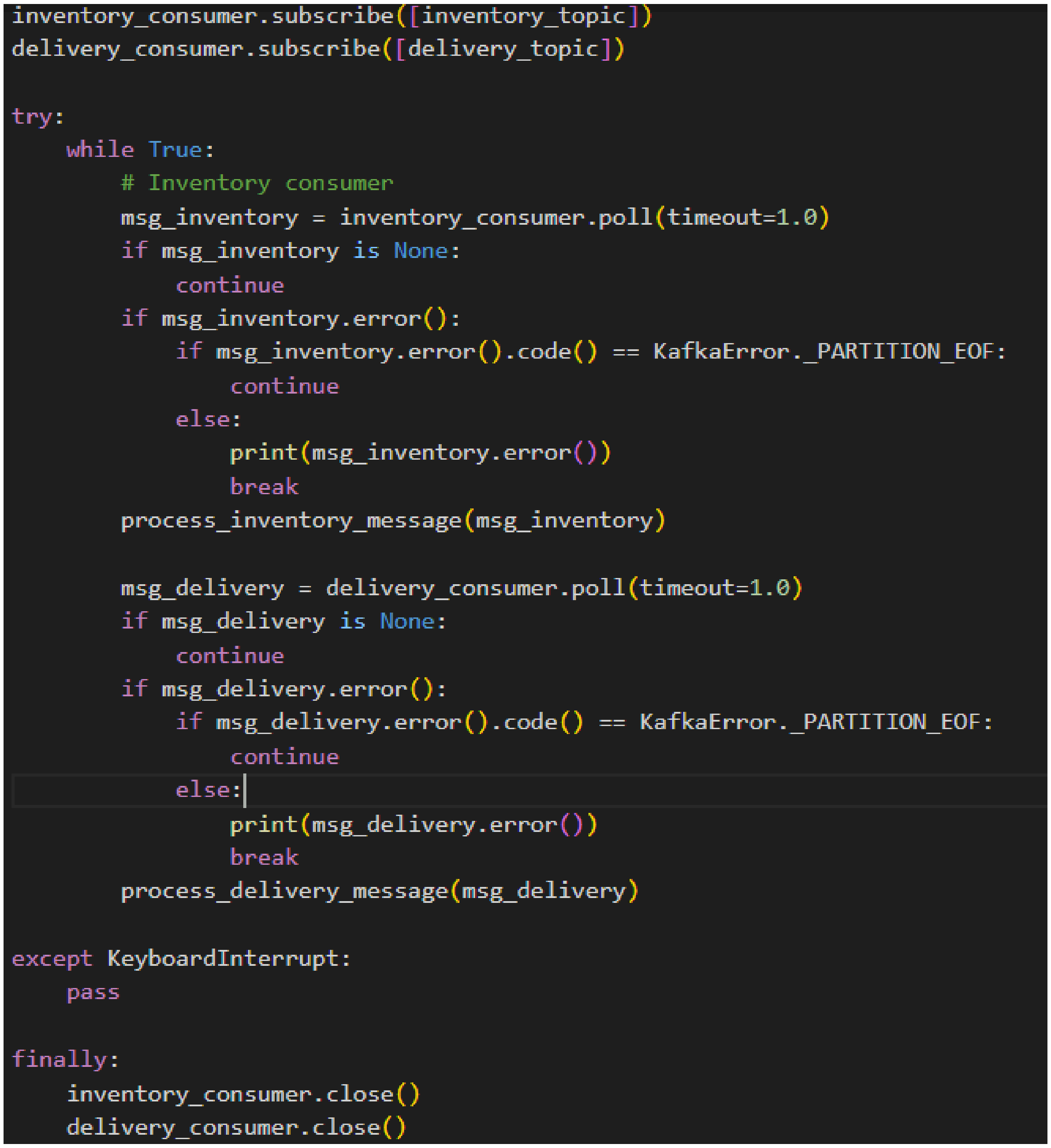
2. Kafka Consumers Implementation:

Consumer 1: Inventory Data Consumer:
Engineered a Kafka consumer application tailored for inventory data management.
Configured the consumer to actively monitor messages tagged with the "inventory" type.
Implemented intricate logic to seamlessly process messages, facilitating seamless updates to inventory databases or systems.

Consumer 2: Delivery Data Consumer:
Established a Kafka consumer tasked with overseeing delivery operations.
Configured the consumer to actively monitor messages labeled with the "delivery" type.
Engineered functionalities enabling delivery scheduling, status updates, and customer notifications in response to incoming messages.



```python
bootstrap_servers = 'localhost:9092'

# Kafka topics
inventory_topic = 'inventory_orders'
delivery_topic = 'delivery_orders'

# Kafka consumer group
consumer_group = 'ecommerce_consumers'

# Kafka consumer configurations
consumer_config = {
    'bootstrap.servers': bootstrap_servers,
    'group.id': consumer_group,
    'auto.offset.reset': 'earliest'
}

inventory_consumer = Consumer(consumer_config)
delivery_consumer = Consumer(consumer_config)

def process_inventory_message(message):
    try:
        order = json.loads(message.value())
        # Process inventory message, update inventory database/system accordingly
        print("Processing inventory order:", order)
    except json.JSONDecodeError as e:
        print("Error decoding JSON:", e)

def process_delivery_message(message):
    try:
        order = json.loads(message.value())
        # Perform delivery action: schedule deliveries, update status, notify customers
        print("Processing delivery order:", order)
    except json.JSONDecodeError as e:
        print("Error decoding JSON:", e)
```



```python
inventory_consumer.subscribe([inventory_topic])
delivery_consumer.subscribe([delivery_topic])

try:
    while True:
        # Inventory consumer
        msg_inventory = inventory_consumer.poll(timeout=1.0)
        if msg_inventory is None:
            continue
        if msg_inventory.error():
            if msg_inventory.error().code() == KafkaError._PARTITION_EOF:
                continue
            else:
                print(msg_inventory.error())
                break
        process_inventory_message(msg_inventory)

        msg_delivery = delivery_consumer.poll(timeout=1.0)
        if msg_delivery is None:
            continue
        if msg_delivery.error():
            if msg_delivery.error().code() == KafkaError._PARTITION_EOF:
                continue
            else:
                print(msg_delivery.error())
                break
        process_delivery_message(msg_delivery)

except KeyboardInterrupt:
    pass

finally:
    inventory_consumer.close()
    delivery_consumer.close()
```