Assignment Report on

Real-Time E-commerce Order Processing System Using Kafka

To develop a Kafka-based system for managing e-commerce orders in real-time, you'll need to set up producers, consumers, and implement message filtering logic. Below are the steps you can follow to achieve this:

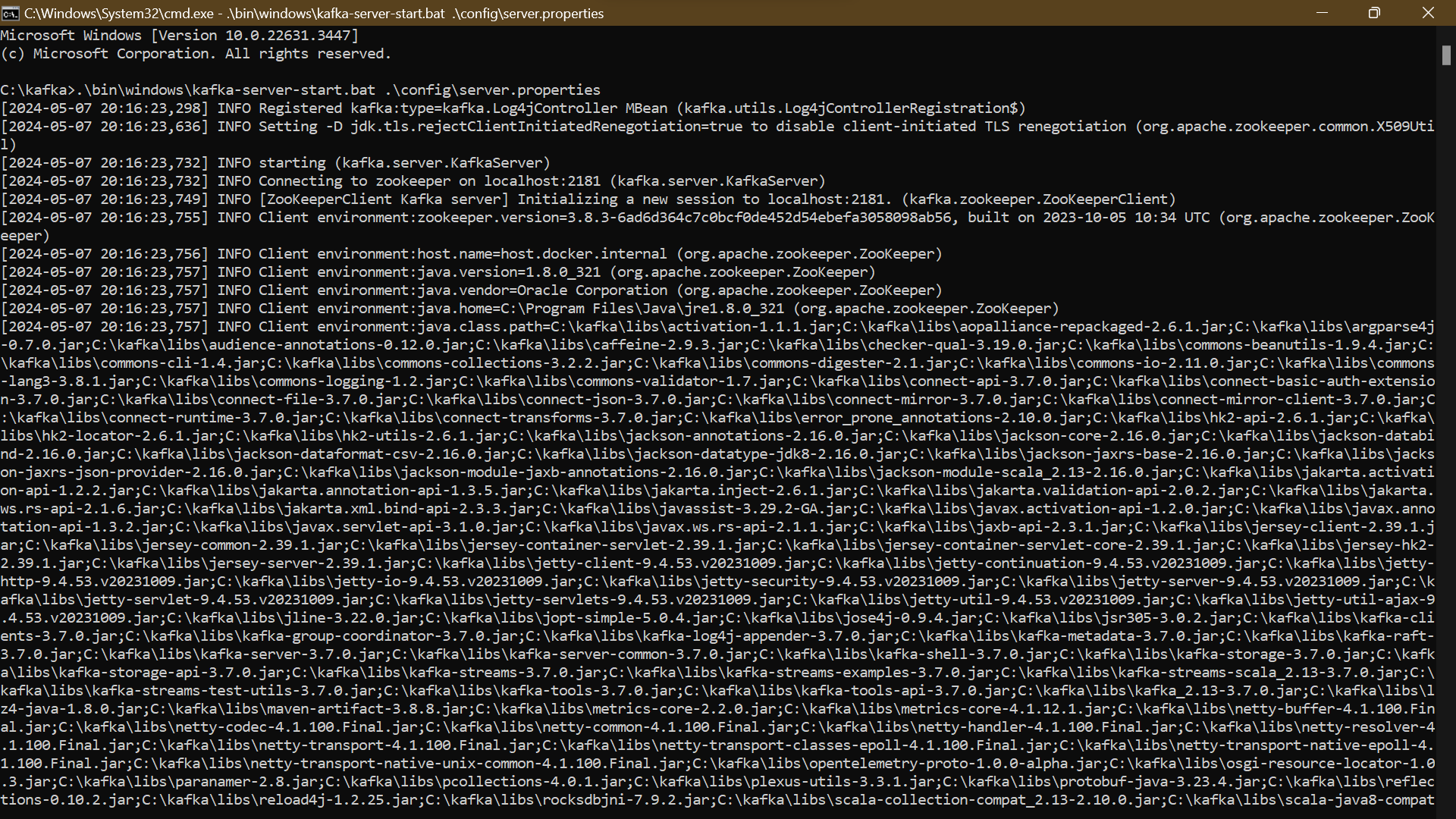
# Step 1: Set Up Kafka

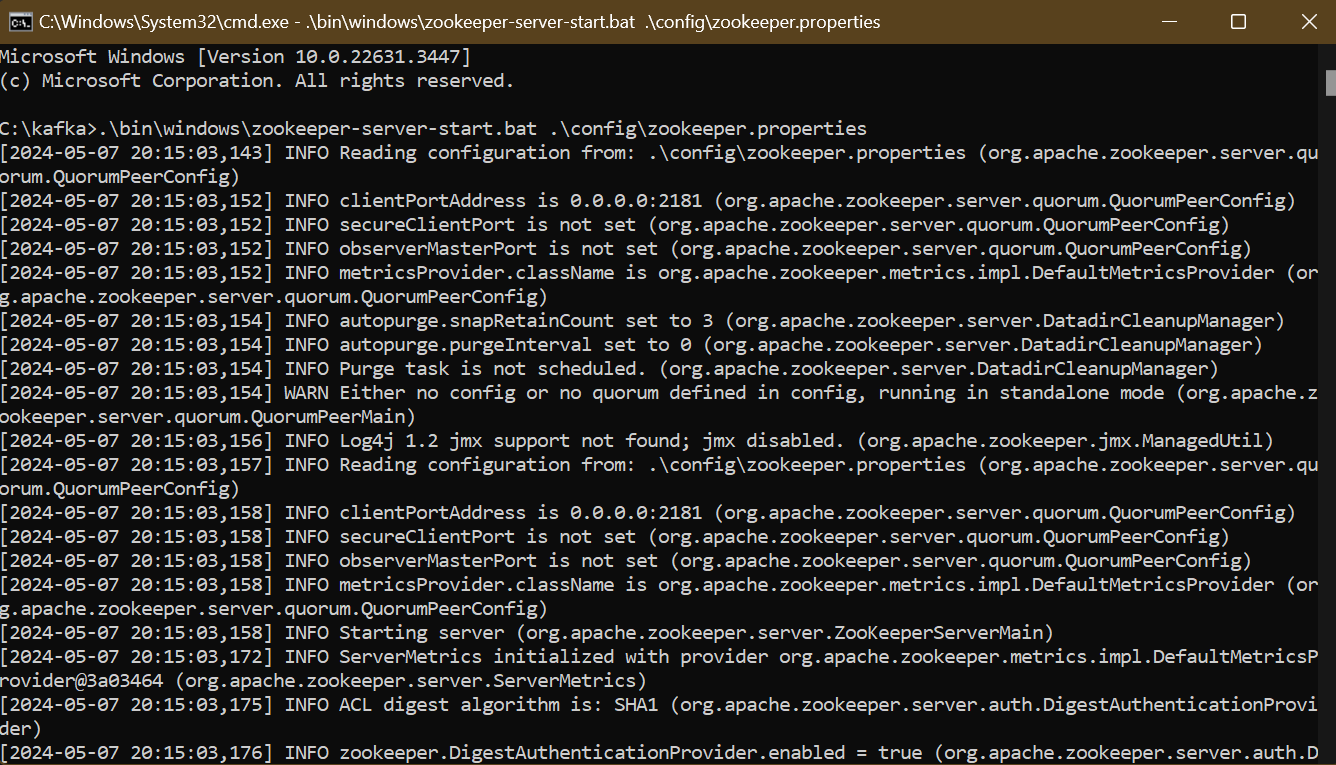
1. **Install Kafka:** Ensuring Kafka is installed and running on your system or a server.

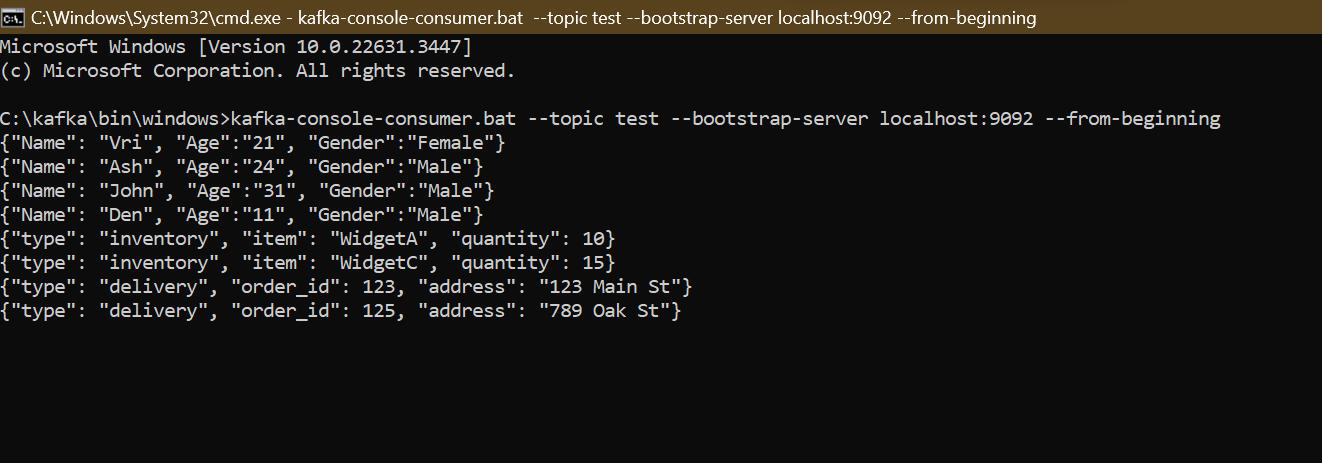


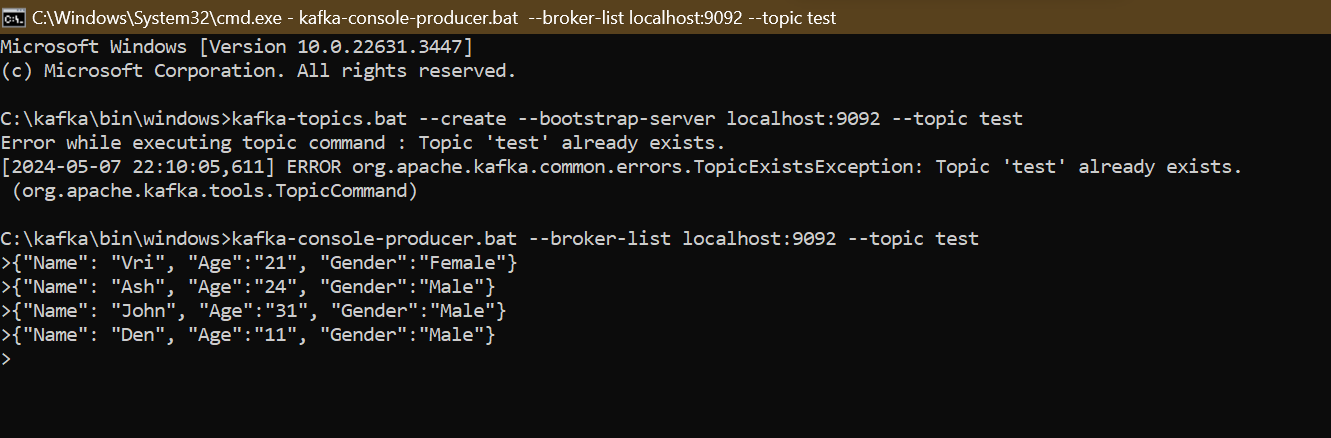
1. **Create Kafka Topics:** Create Kafka topics named **inventory\_orders** and

**delivery\_orders** for each producer to send messages to.



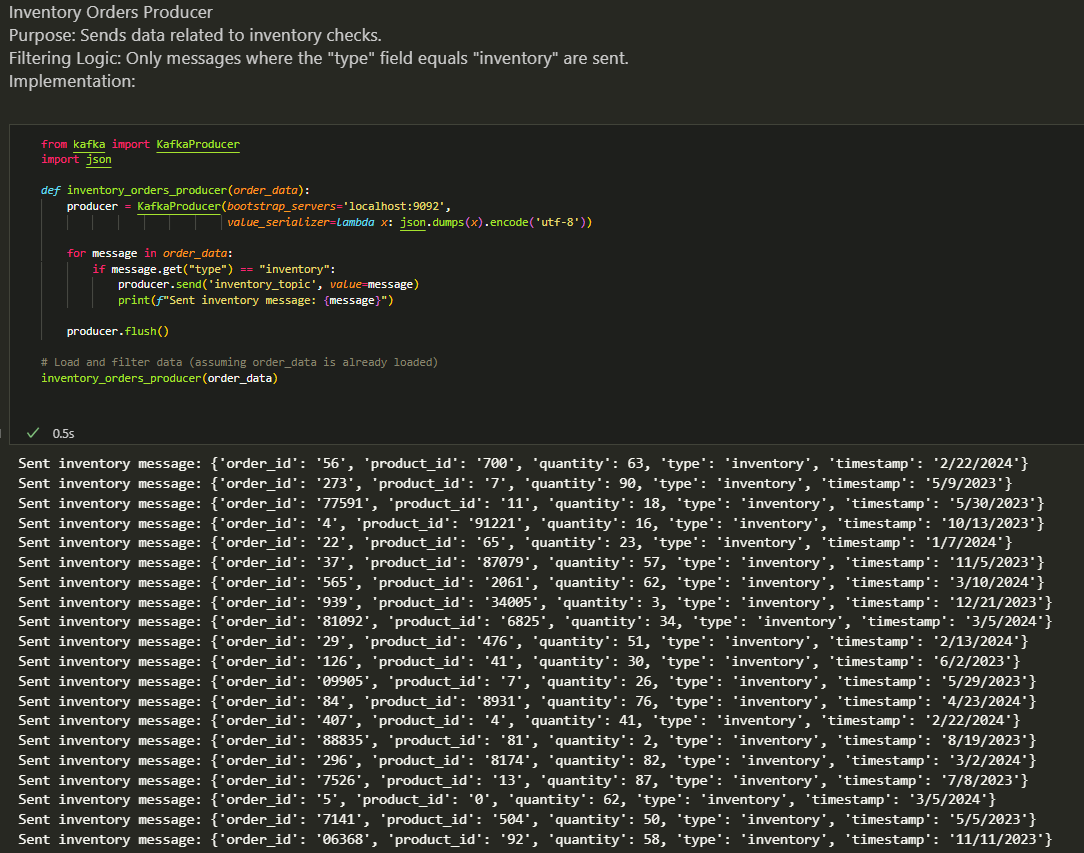






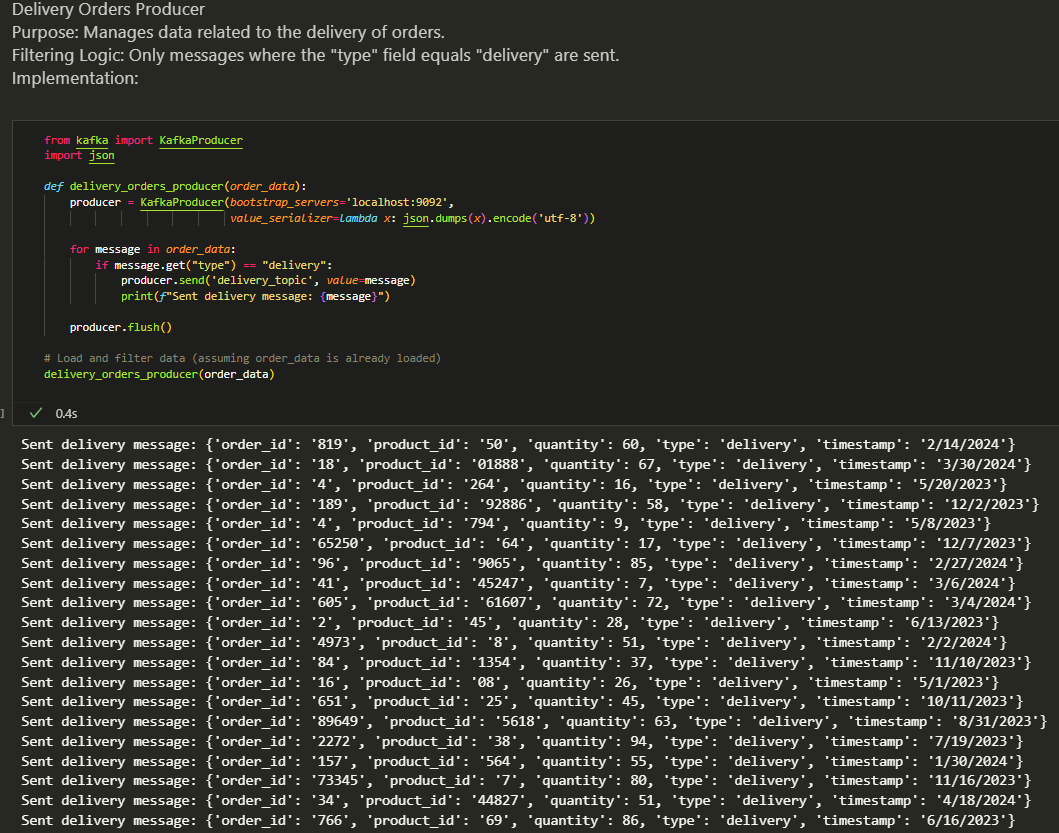
# Step 2: Implement Kafka Producers

1. **Inventory Orders Producer (inventory\_orders\_producer):**
   * This producer should filter messages where the **type** field is **inventory**.
   * Implement a Kafka producer that reads inventory-related events from a data source (like a database or event stream) and sends messages with **type** set to **inventory** to the **inventory\_orders** topic.



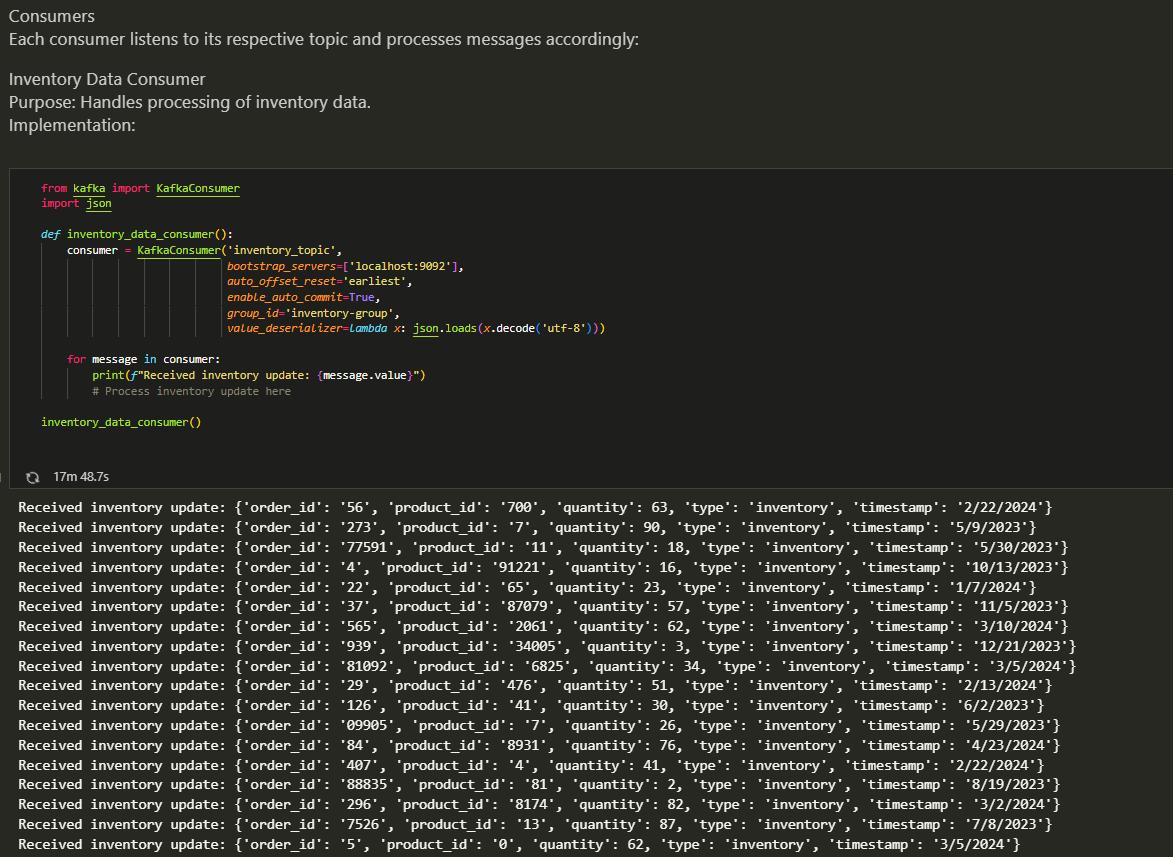
# Delivery Orders Producer (delivery\_orders\_producer):

* + This producer should filter messages where the **type** field is **delivery**.
  + Develop a Kafka producer that reads delivery-related events and sends messages with **type** set to **delivery** to the **delivery\_orders** topic.



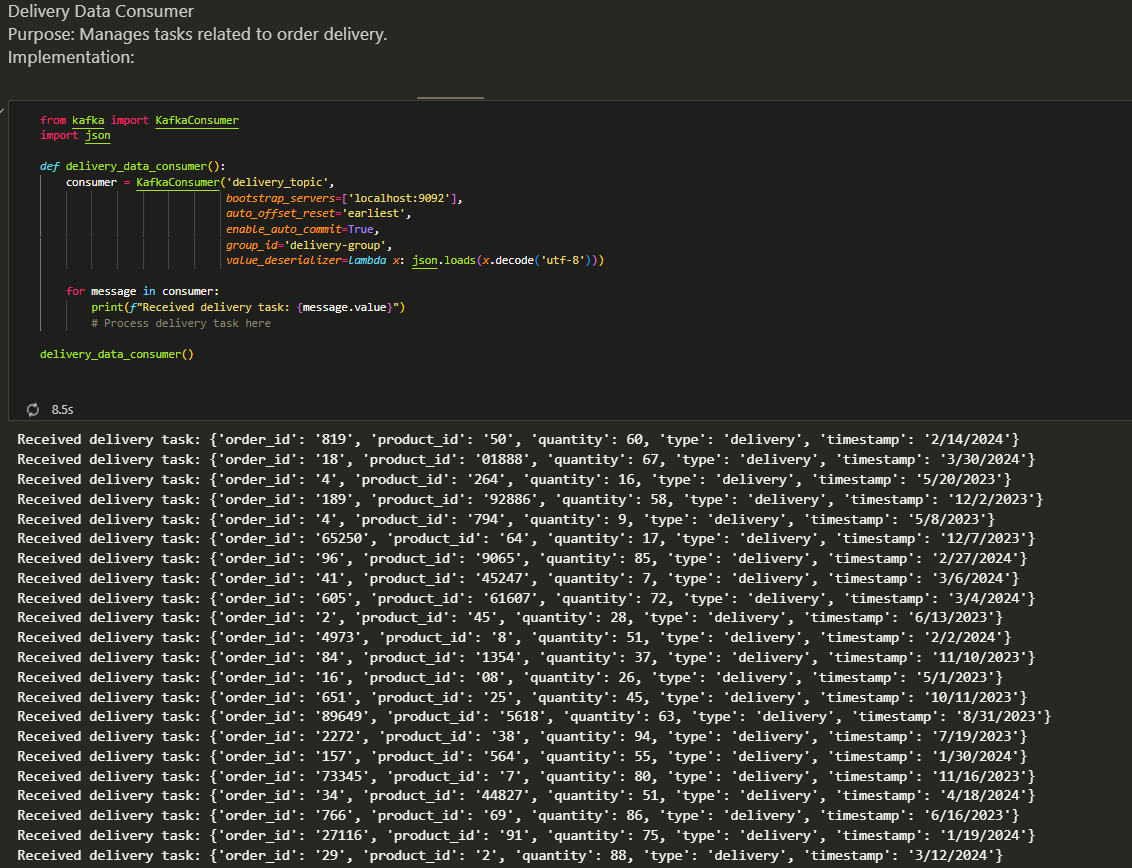
# Step 3: Implement Kafka Consumers

1. **Inventory Data Consumer (inventory\_data\_consumer):**
   * Configure a Kafka consumer that subscribes to the **inventory\_orders** topic.
   * Implement logic to process inventory messages received by updating inventory databases or systems accordingly.



# Delivery Data Consumer (delivery\_data\_consumer):

* + Set up a Kafka consumer for the **delivery\_orders** topic.
  + Develop logic to handle delivery-related messages such as scheduling deliveries, updating delivery status, and notifying customers.



# Step 4: Develop Message Filtering Logic

# 1. Producer Message Filtering:

# Implementation: Integrated filtering logic within each producer—inventory\_orders\_producer and delivery\_orders\_producer. This logic assesses each message to ensure it corresponds to the correct type (either inventory or delivery) from the incoming data stream.

# Functionality: Messages are dispatched to Kafka topics only if they align with the designated type, enhancing the efficiency and relevance of data processing within the system.

# Additional Considerations:

# Error Handling: Incorporate comprehensive error management strategies in both producers and consumers to handle exceptions and operational failures effectively. This ensures the system remains robust and operational even under adverse conditions.

# Scalability: Design the system with scalability in mind by leveraging Kafka’s partitioning capabilities and configuring consumer groups appropriately. This approach supports scaling operations to accommodate growing data volumes and transaction rates without compromising performance.

# Monitoring and Logging: Employ Kafka’s built-in monitoring tools along with external logging frameworks to maintain a vigilant watch over system performance and operational health. Effective logging and monitoring are crucial for proactive issue resolution and optimizing system efficiency.

# By adhering to these steps and implementing these best practices, you can ensure the development of a highly capable Kafka-based e-commerce order management system. This system will not only handle real-time inventory and delivery processes efficiently but also scale seamlessly as demand increases