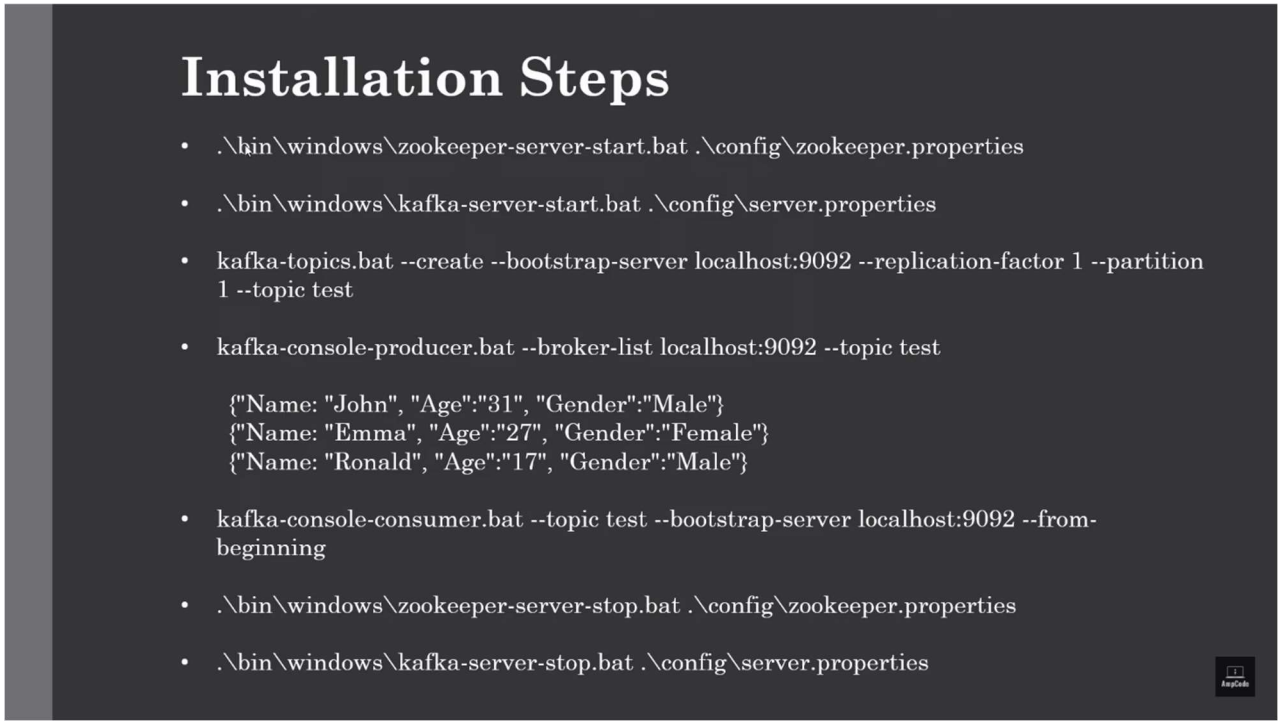
Assignment Report on

Real-Time E-commerce Order Processing System Using Kafka

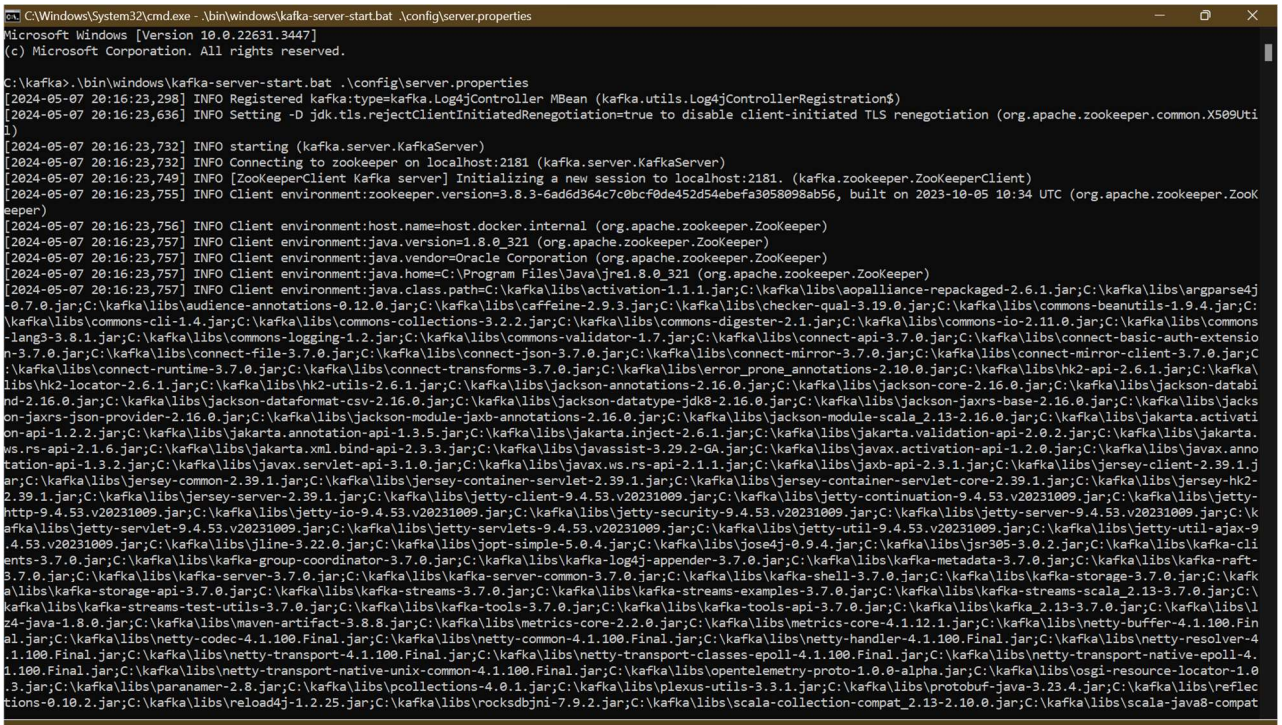
To develop a Kafka-based system for managing e-commerce orders in real-time, you'll need to set up producers, consumers, and implement message filtering logic. Below are the steps you can follow to achieve this:

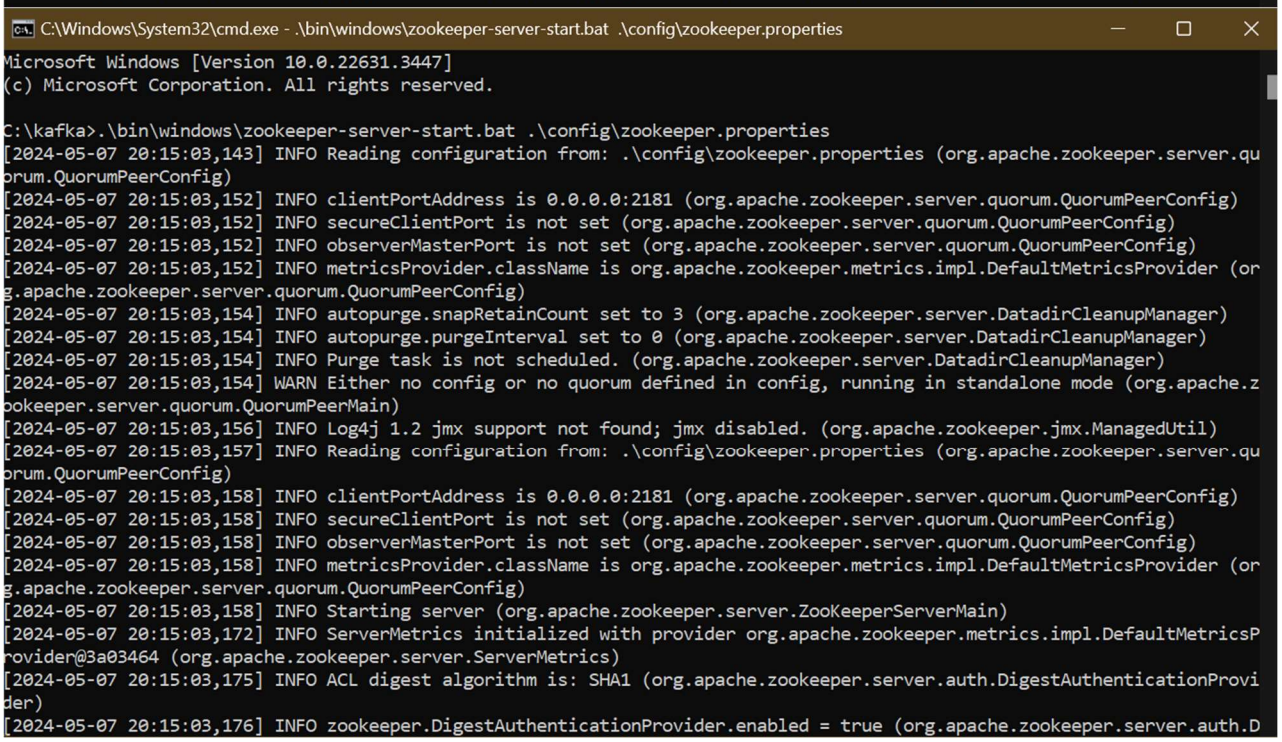
Step 1: Set Up Kafka

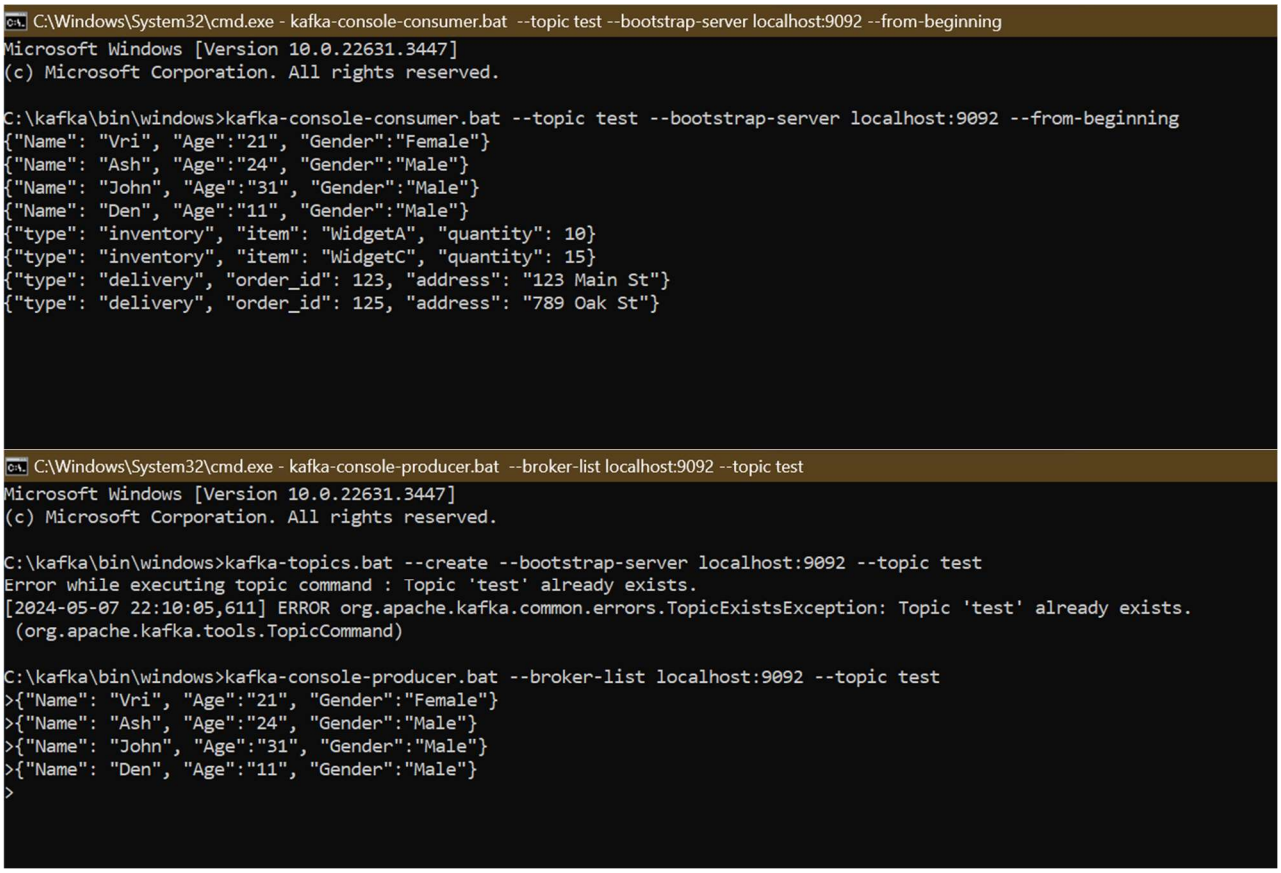
1. **Install Kafka:** Ensuring Kafka is installed and running on your system or a server.



2. **Create Kafka Topics:** Create Kafka topics named **inventory_orders** and **delivery_orders** for each producer to send messages to.


C:\Windows\System32\cmd.exe - .\bin\windows\kafka-server-start.bat .\config\server.properties
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.
C:\kafka>.\bin\windows\kafka-server-start.bat .\config\server.properties
[2024-05-07 20:16:23,298] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2024-05-07 20:16:23,636] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Uti
l)
[2024-05-07 20:16:23,732] INFO starting (kafka.server.KafkaServer)
[2024-05-07 20:16:23,732] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2024-05-07 20:16:23,749] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2024-05-07 20:16:23,755] INFO Client environment:zookeeper.version=3.8.3-6ad6d364c7c0bcf0de452d54ebefa3058098ab56, built on 2023-10-05 10:34 UTC (org.apache.zookeeper.ZooK
eeper)
[2024-05-07 20:16:23,756] INFO Client environment:host.name=host.docker.internal (org.apache.zookeeper.ZooKeeper)
[2024-05-07 20:16:23,757] INFO Client environment:java.version=1.8.0_321 (org.apache.zookeeper.ZooKeeper)
[2024-05-07 20:16:23,757] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2024-05-07 20:16:23,757] INFO Client environment:java.home=C:\Program Files\Java\jre1.8.0_321 (org.apache.zookeeper.ZooKeeper)
[2024-05-07 20:16:23,757] INFO Client environment:java.class.path=C:\kafka\libs\activation-1.1.1.jar;C:\kafka\libs\aopalliance-repackaged-2.6.1.jar;C:\kafka\libs\argparse4j
-0.7.0.jar;C:\kafka\libs\audience-annotations-0.12.0.jar;C:\kafka\libs\caffeine-2.9.3.jar;C:\kafka\libs\checker-qual-3.19.0.jar;C:\kafka\libs\commons-beanutils-1.9.4.jar;C:
\kafka\libs\commons-cli-1.4.jar;C:\kafka\libs\commons-collections-3.2.2.jar;C:\kafka\libs\commons-digester-2.1.jar;C:\kafka\libs\commons-io-2.11.0.jar;C:\kafka\libs\commons
-lang3-3.8.1.jar;C:\kafka\libs\commons-logging-1.2.jar;C:\kafka\libs\commons-validator-1.7.jar;C:\kafka\libs\connect-api-3.7.0.jar;C:\kafka\libs\connect-basic-auth-extensio
n-3.7.0.jar;C:\kafka\libs\connect-file-3.7.0.jar;C:\kafka\libs\connect-json-3.7.0.jar;C:\kafka\libs\connect-mirror-3.7.0.jar;C:\kafka\libs\connect-mirror-client-3.7.0.jar;C
:\kafka\libs\connect-runtime-3.7.0.jar;C:\kafka\libs\connect-transforms-3.7.0.jar;C:\kafka\libs\error_prone_annotations-2.10.0.jar;C:\kafka\libs\hk2-api-2.6.1.jar;C:\kafka\
libs\hk2-locator-2.6.1.jar;C:\kafka\libs\hk2-utils-2.6.1.jar;C:\kafka\libs\jackson-annotations-2.16.0.jar;C:\kafka\libs\jackson-core-2.16.0.jar;C:\kafka\libs\jackson-databi
nd-2.16.0.jar;C:\kafka\libs\jackson-dataformat-csv-2.16.0.jar;C:\kafka\libs\jackson-datatype-jdk8-2.16.0.jar;C:\kafka\libs\jackson-jaxrs-base-2.16.0.jar;C:\kafka\libs\jacks
on-jaxrs-json-provider-2.16.0.jar;C:\kafka\libs\jackson-module-jaxb-annotations-2.16.0.jar;C:\kafka\libs\jackson-module-scala_2.13-2.16.0.jar;C:\kafka\libs\jakarta.activati
on-api-1.2.2.jar;C:\kafka\libs\jakarta.annotation-api-1.3.5.jar;C:\kafka\libs\jakarta.inject-2.6.1.jar;C:\kafka\libs\jakarta.validation-api-2.0.2.jar;C:\kafka\libs\jakarta.
ws.rs-api-2.1.6.jar;C:\kafka\libs\jakarta.xml.bind-api-2.3.3.jar;C:\kafka\libs\javassist-3.29.2-GA.jar;C:\kafka\libs\javax.activation-api-1.2.0.jar;C:\kafka\libs\javax.anno
tation-api-1.3.2.jar;C:\kafka\libs\javax.servlet-api-3.1.0.jar;C:\kafka\libs\javax.ws.rs-api-2.1.1.jar;C:\kafka\libs\jaxb-api-2.3.1.jar;C:\kafka\libs\jersey-client-2.39.1.j
ar;C:\kafka\libs\jersey-common-2.39.1.jar;C:\kafka\libs\jersey-container-servlet-2.39.1.jar;C:\kafka\libs\jersey-container-servlet-core-2.39.1.jar;C:\kafka\libs\jersey-hk2-
2.39.1.jar;C:\kafka\libs\jersey-server-2.39.1.jar;C:\kafka\libs\jetty-client-9.4.53.v20231009.jar;C:\kafka\libs\jetty-continuation-9.4.53.v20231009.jar;C:\kafka\libs\jetty-
http-9.4.53.v20231009.jar;C:\kafka\libs\jetty-io-9.4.53.v20231009.jar;C:\kafka\libs\jetty-security-9.4.53.v20231009.jar;C:\kafka\libs\jetty-server-9.4.53.v20231009.jar;C:\k
afka\libs\jetty-servlet-9.4.53.v20231009.jar;C:\kafka\libs\jetty-servlets-9.4.53.v20231009.jar;C:\kafka\libs\jetty-util-9.4.53.v20231009.jar;C:\kafka\libs\jetty-util-ajax-9
.4.53.v20231009.jar;C:\kafka\libs\jline-3.22.0.jar;C:\kafka\libs\jopt-simple-5.0.4.jar;C:\kafka\libs\jose4j-0.9.4.jar;C:\kafka\libs\jsr305-3.0.2.jar;C:\kafka\libs\kafka-cli
ents-3.7.0.jar;C:\kafka\libs\kafka-group-coordinator-3.7.0.jar;C:\kafka\libs\kafka-log4j-appender-3.7.0.jar;C:\kafka\libs\kafka-metadata-3.7.0.jar;C:\kafka\libs\kafka-raft-
3.7.0.jar;C:\kafka\libs\kafka-server-3.7.0.jar;C:\kafka\libs\kafka-server-common-3.7.0.jar;C:\kafka\libs\kafka-shell-3.7.0.jar;C:\kafka\libs\kafka-storage-3.7.0.jar;C:\kafk
a\libs\kafka-storage-api-3.7.0.jar;C:\kafka\libs\kafka-streams-3.7.0.jar;C:\kafka\libs\kafka-streams-examples-3.7.0.jar;C:\kafka\libs\kafka-streams-scala_2.13-3.7.0.jar;C:\
kafka\libs\kafka-streams-test-utils-3.7.0.jar;C:\kafka\libs\kafka-tools-3.7.0.jar;C:\kafka\libs\kafka-tools-api-3.7.0.jar;C:\kafka\libs\kafka_2.13-3.7.0.jar;C:\kafka\libs\l
z4-java-1.8.0.jar;C:\kafka\libs\maven-artifact-3.8.8.jar;C:\kafka\libs\metrics-core-2.2.0.jar;C:\kafka\libs\metrics-core-4.1.12.1.jar;C:\kafka\libs\netty-buffer-4.1.100.Fin
al.jar;C:\kafka\libs\netty-codec-4.1.100.Final.jar;C:\kafka\libs\netty-common-4.1.100.Final.jar;C:\kafka\libs\netty-handler-4.1.100.Final.jar;C:\kafka\libs\netty-resolver-4
.1.100.Final.jar;C:\kafka\libs\netty-transport-4.1.100.Final.jar;C:\kafka\libs\netty-transport-classes-epoll-4.1.100.Final.jar;C:\kafka\libs\netty-transport-native-epoll-4.
1.100.Final.jar;C:\kafka\libs\netty-transport-native-unix-common-4.1.100.Final.jar;C:\kafka\libs\opentelemetry-proto-1.0.0-alpha.jar;C:\kafka\libs\osgi-resource-locator-1.0
.3.jar;C:\kafka\libs\paranamer-2.8.jar;C:\kafka\libs\pcollections-4.0.1.jar;C:\kafka\libs\plexus-utils-3.3.1.jar;C:\kafka\libs\protobuf-java-3.23.4.jar;C:\kafka\libs\reflec
tions-0.10.2.jar;C:\kafka\libs\reload4j-1.2.25.jar;C:\kafka\libs\rocksdbjni-7.9.2.jar;C:\kafka\libs\scala-collection-compat_2.13-2.10.0.jar;C:\kafka\libs\scala-java8-compat


C:\Windows\System32\cmd.exe - .\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.
C:\kafka>.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
[2024-05-07 20:15:03,143] INFO Reading configuration from: .\config\zookeeper.properties (org.apache.zookeeper.server.qu
orum.QuorumPeerConfig)
[2024-05-07 20:15:03,152] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-07 20:15:03,152] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-07 20:15:03,152] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-07 20:15:03,152] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (or
g.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-07 20:15:03,154] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-05-07 20:15:03,154] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-05-07 20:15:03,154] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-05-07 20:15:03,154] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.z
ookeeper.server.quorum.QuorumPeerMain)
[2024-05-07 20:15:03,156] INFO Log4j 1.2 jmx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2024-05-07 20:15:03,157] INFO Reading configuration from: .\config\zookeeper.properties (org.apache.zookeeper.server.qu
orum.QuorumPeerConfig)
[2024-05-07 20:15:03,158] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-07 20:15:03,158] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-07 20:15:03,158] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-07 20:15:03,158] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (or
g.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-07 20:15:03,158] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2024-05-07 20:15:03,172] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsP
rovider@3a03464 (org.apache.zookeeper.server.ServerMetrics)
[2024-05-07 20:15:03,175] INFO ACL digest algorithm is: SHA1 (org.apache.zookeeper.server.auth.DigestAuthenticationProvi
der)
[2024-05-07 20:15:03,176] INFO zookeeper.DigestAuthenticationProvider.enabled = true (org.apache.zookeeper.server.auth.D



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.

Inventory Orders Producer
Purpose: Sends data related to inventory checks.
Filtering Logic: Only messages where the "type" field equals "inventory" are sent.
Implementation:

```python
from kafka import KafkaProducer
import json

def inventory_orders_producer(order_data):
    producer = KafkaProducer(bootstrap_servers='localhost:9092',
                             value_serializer=lambda x: json.dumps(x).encode('utf-8'))

    for message in order_data:
        if message.get("type") == "inventory":
            producer.send('inventory_topic', value=message)
            print(f"Sent inventory message: {message}")

    producer.flush()

# Load and filter data (assuming order_data is already loaded)
inventory_orders_producer(order_data)
```

✓ 0.5s

```
Sent inventory message: {'order_id': '56', 'product_id': '700', 'quantity': 63, 'type': 'inventory', 'timestamp': '2/22/2024'}
Sent inventory message: {'order_id': '273', 'product_id': '7', 'quantity': 90, 'type': 'inventory', 'timestamp': '5/9/2023'}
Sent inventory message: {'order_id': '77591', 'product_id': '11', 'quantity': 18, 'type': 'inventory', 'timestamp': '5/30/2023'}
Sent inventory message: {'order_id': '4', 'product_id': '91221', 'quantity': 16, 'type': 'inventory', 'timestamp': '10/13/2023'}
Sent inventory message: {'order_id': '22', 'product_id': '65', 'quantity': 23, 'type': 'inventory', 'timestamp': '1/7/2024'}
Sent inventory message: {'order_id': '37', 'product_id': '87079', 'quantity': 57, 'type': 'inventory', 'timestamp': '11/5/2023'}
Sent inventory message: {'order_id': '565', 'product_id': '2061', 'quantity': 62, 'type': 'inventory', 'timestamp': '3/10/2024'}
Sent inventory message: {'order_id': '939', 'product_id': '34005', 'quantity': 3, 'type': 'inventory', 'timestamp': '12/21/2023'}
Sent inventory message: {'order_id': '81092', 'product_id': '6825', 'quantity': 34, 'type': 'inventory', 'timestamp': '3/5/2024'}
Sent inventory message: {'order_id': '29', 'product_id': '476', 'quantity': 51, 'type': 'inventory', 'timestamp': '2/13/2024'}
Sent inventory message: {'order_id': '126', 'product_id': '41', 'quantity': 30, 'type': 'inventory', 'timestamp': '6/2/2023'}
Sent inventory message: {'order_id': '09905', 'product_id': '7', 'quantity': 26, 'type': 'inventory', 'timestamp': '5/29/2023'}
Sent inventory message: {'order_id': '84', 'product_id': '8931', 'quantity': 76, 'type': 'inventory', 'timestamp': '4/23/2024'}
Sent inventory message: {'order_id': '407', 'product_id': '4', 'quantity': 41, 'type': 'inventory', 'timestamp': '2/22/2024'}
Sent inventory message: {'order_id': '88835', 'product_id': '81', 'quantity': 2, 'type': 'inventory', 'timestamp': '8/19/2023'}
Sent inventory message: {'order_id': '296', 'product_id': '8174', 'quantity': 82, 'type': 'inventory', 'timestamp': '3/2/2024'}
Sent inventory message: {'order_id': '7526', 'product_id': '13', 'quantity': 87, 'type': 'inventory', 'timestamp': '7/8/2023'}
Sent inventory message: {'order_id': '5', 'product_id': '0', 'quantity': 62, 'type': 'inventory', 'timestamp': '3/5/2024'}
Sent inventory message: {'order_id': '7141', 'product_id': '504', 'quantity': 50, 'type': 'inventory', 'timestamp': '5/5/2023'}
Sent inventory message: {'order_id': '06368', 'product_id': '92', 'quantity': 58, 'type': 'inventory', 'timestamp': '11/11/2023'}
```

2. **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.

Delivery Orders Producer
Purpose: Manages data related to the delivery of orders.
Filtering Logic: Only messages where the "type" field equals "delivery" are sent.
Implementation:

```python
from kafka import KafkaProducer
import json

def delivery_orders_producer(order_data):
    producer = KafkaProducer(bootstrap_servers='localhost:9092',
                             value_serializer=lambda x: json.dumps(x).encode('utf-8'))

    for message in order_data:
        if message.get("type") == "delivery":
            producer.send('delivery_topic', value=message)
            print(f"Sent delivery message: {message}")

    producer.flush()

# Load and filter data (assuming order_data is already loaded)
delivery_orders_producer(order_data)
```

✓ 0.4s

```
Sent delivery message: {'order_id': '819', 'product_id': '50', 'quantity': 60, 'type': 'delivery', 'timestamp': '2/14/2024'}
Sent delivery message: {'order_id': '18', 'product_id': '01888', 'quantity': 67, 'type': 'delivery', 'timestamp': '3/30/2024'}
Sent delivery message: {'order_id': '4', 'product_id': '264', 'quantity': 16, 'type': 'delivery', 'timestamp': '5/20/2023'}
Sent delivery message: {'order_id': '189', 'product_id': '92886', 'quantity': 58, 'type': 'delivery', 'timestamp': '12/2/2023'}
Sent delivery message: {'order_id': '4', 'product_id': '794', 'quantity': 9, 'type': 'delivery', 'timestamp': '5/8/2023'}
Sent delivery message: {'order_id': '65250', 'product_id': '64', 'quantity': 17, 'type': 'delivery', 'timestamp': '12/7/2023'}
Sent delivery message: {'order_id': '96', 'product_id': '9065', 'quantity': 85, 'type': 'delivery', 'timestamp': '2/27/2024'}
Sent delivery message: {'order_id': '41', 'product_id': '45247', 'quantity': 7, 'type': 'delivery', 'timestamp': '3/6/2024'}
Sent delivery message: {'order_id': '605', 'product_id': '61607', 'quantity': 72, 'type': 'delivery', 'timestamp': '3/4/2024'}
Sent delivery message: {'order_id': '2', 'product_id': '45', 'quantity': 28, 'type': 'delivery', 'timestamp': '6/13/2023'}
Sent delivery message: {'order_id': '4973', 'product_id': '8', 'quantity': 51, 'type': 'delivery', 'timestamp': '2/2/2024'}
Sent delivery message: {'order_id': '84', 'product_id': '1354', 'quantity': 37, 'type': 'delivery', 'timestamp': '11/10/2023'}
Sent delivery message: {'order_id': '16', 'product_id': '08', 'quantity': 26, 'type': 'delivery', 'timestamp': '5/1/2023'}
Sent delivery message: {'order_id': '651', 'product_id': '25', 'quantity': 45, 'type': 'delivery', 'timestamp': '10/11/2023'}
Sent delivery message: {'order_id': '89649', 'product_id': '5618', 'quantity': 63, 'type': 'delivery', 'timestamp': '8/31/2023'}
Sent delivery message: {'order_id': '2272', 'product_id': '38', 'quantity': 94, 'type': 'delivery', 'timestamp': '7/19/2023'}
Sent delivery message: {'order_id': '157', 'product_id': '564', 'quantity': 55, 'type': 'delivery', 'timestamp': '1/30/2024'}
Sent delivery message: {'order_id': '73345', 'product_id': '7', 'quantity': 80, 'type': 'delivery', 'timestamp': '11/16/2023'}
Sent delivery message: {'order_id': '34', 'product_id': '44827', 'quantity': 51, 'type': 'delivery', 'timestamp': '4/18/2024'}
Sent delivery message: {'order_id': '766', 'product_id': '69', 'quantity': 86, 'type': 'delivery', 'timestamp': '6/16/2023'}
```

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.

Consumers
Each consumer listens to its respective topic and processes messages accordingly:

Inventory Data Consumer
Purpose: Handles processing of inventory data.
Implementation:

```python
from kafka import KafkaConsumer
import json

def inventory_data_consumer():
    consumer = KafkaConsumer('inventory_topic',
                             bootstrap_servers=['localhost:9092'],
                             auto_offset_reset='earliest',
                             enable_auto_commit=True,
                             group_id='inventory-group',
                             value_deserializer=lambda x: json.loads(x.decode('utf-8')))

    for message in consumer:
        print(f"Received inventory update: {message.value}")
        # Process inventory update here

inventory_data_consumer()
```

17m 48.7s

```
Received inventory update: {'order_id': '56', 'product_id': '700', 'quantity': 63, 'type': 'inventory', 'timestamp': '2/22/2024'}
Received inventory update: {'order_id': '273', 'product_id': '7', 'quantity': 90, 'type': 'inventory', 'timestamp': '5/9/2023'}
Received inventory update: {'order_id': '77591', 'product_id': '11', 'quantity': 18, 'type': 'inventory', 'timestamp': '5/30/2023'}
Received inventory update: {'order_id': '4', 'product_id': '91221', 'quantity': 16, 'type': 'inventory', 'timestamp': '10/13/2023'}
Received inventory update: {'order_id': '22', 'product_id': '65', 'quantity': 23, 'type': 'inventory', 'timestamp': '1/7/2024'}
Received inventory update: {'order_id': '37', 'product_id': '87079', 'quantity': 57, 'type': 'inventory', 'timestamp': '11/5/2023'}
Received inventory update: {'order_id': '565', 'product_id': '2061', 'quantity': 62, 'type': 'inventory', 'timestamp': '3/10/2024'}
Received inventory update: {'order_id': '939', 'product_id': '34005', 'quantity': 3, 'type': 'inventory', 'timestamp': '12/21/2023'}
Received inventory update: {'order_id': '81092', 'product_id': '6825', 'quantity': 34, 'type': 'inventory', 'timestamp': '3/5/2024'}
Received inventory update: {'order_id': '29', 'product_id': '476', 'quantity': 51, 'type': 'inventory', 'timestamp': '2/13/2024'}
Received inventory update: {'order_id': '126', 'product_id': '41', 'quantity': 30, 'type': 'inventory', 'timestamp': '6/2/2023'}
Received inventory update: {'order_id': '09905', 'product_id': '7', 'quantity': 26, 'type': 'inventory', 'timestamp': '5/29/2023'}
Received inventory update: {'order_id': '84', 'product_id': '8931', 'quantity': 76, 'type': 'inventory', 'timestamp': '4/23/2024'}
Received inventory update: {'order_id': '407', 'product_id': '4', 'quantity': 41, 'type': 'inventory', 'timestamp': '2/22/2024'}
Received inventory update: {'order_id': '88835', 'product_id': '81', 'quantity': 2, 'type': 'inventory', 'timestamp': '8/19/2023'}
Received inventory update: {'order_id': '296', 'product_id': '8174', 'quantity': 82, 'type': 'inventory', 'timestamp': '3/2/2024'}
Received inventory update: {'order_id': '7526', 'product_id': '13', 'quantity': 87, 'type': 'inventory', 'timestamp': '7/8/2023'}
Received inventory update: {'order_id': '5', 'product_id': '0', 'quantity': 62, 'type': 'inventory', 'timestamp': '3/5/2024'}
```

2. **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.

Delivery Data Consumer
Purpose: Manages tasks related to order delivery.
Implementation:

```python
from kafka import KafkaConsumer
import json

def delivery_data_consumer():
    consumer = KafkaConsumer('delivery_topic',
                             bootstrap_servers=['localhost:9092'],
                             auto_offset_reset='earliest',
                             enable_auto_commit=True,
                             group_id='delivery-group',
                             value_deserializer=lambda x: json.loads(x.decode('utf-8')))

    for message in consumer:
        print(f"Received delivery task: {message.value}")
        # Process delivery task here

delivery_data_consumer()
```

8.5s

```
Received delivery task: {'order_id': '819', 'product_id': '50', 'quantity': 60, 'type': 'delivery', 'timestamp': '2/14/2024'}
Received delivery task: {'order_id': '18', 'product_id': '01888', 'quantity': 67, 'type': 'delivery', 'timestamp': '3/30/2024'}
Received delivery task: {'order_id': '4', 'product_id': '264', 'quantity': 16, 'type': 'delivery', 'timestamp': '5/20/2023'}
Received delivery task: {'order_id': '189', 'product_id': '92886', 'quantity': 58, 'type': 'delivery', 'timestamp': '12/2/2023'}
Received delivery task: {'order_id': '4', 'product_id': '794', 'quantity': 9, 'type': 'delivery', 'timestamp': '5/8/2023'}
Received delivery task: {'order_id': '65250', 'product_id': '64', 'quantity': 17, 'type': 'delivery', 'timestamp': '12/7/2023'}
Received delivery task: {'order_id': '96', 'product_id': '9065', 'quantity': 85, 'type': 'delivery', 'timestamp': '2/27/2024'}
Received delivery task: {'order_id': '41', 'product_id': '45247', 'quantity': 7, 'type': 'delivery', 'timestamp': '3/6/2024'}
Received delivery task: {'order_id': '605', 'product_id': '61607', 'quantity': 72, 'type': 'delivery', 'timestamp': '3/4/2024'}
Received delivery task: {'order_id': '2', 'product_id': '45', 'quantity': 28, 'type': 'delivery', 'timestamp': '6/13/2023'}
Received delivery task: {'order_id': '4973', 'product_id': '8', 'quantity': 51, 'type': 'delivery', 'timestamp': '2/2/2024'}
Received delivery task: {'order_id': '84', 'product_id': '1354', 'quantity': 37, 'type': 'delivery', 'timestamp': '11/10/2023'}
Received delivery task: {'order_id': '16', 'product_id': '08', 'quantity': 26, 'type': 'delivery', 'timestamp': '5/1/2023'}
Received delivery task: {'order_id': '651', 'product_id': '25', 'quantity': 45, 'type': 'delivery', 'timestamp': '10/11/2023'}
Received delivery task: {'order_id': '89649', 'product_id': '5618', 'quantity': 63, 'type': 'delivery', 'timestamp': '8/31/2023'}
Received delivery task: {'order_id': '2272', 'product_id': '38', 'quantity': 94, 'type': 'delivery', 'timestamp': '7/19/2023'}
Received delivery task: {'order_id': '157', 'product_id': '564', 'quantity': 55, 'type': 'delivery', 'timestamp': '1/30/2024'}
Received delivery task: {'order_id': '73345', 'product_id': '7', 'quantity': 80, 'type': 'delivery', 'timestamp': '11/16/2023'}
Received delivery task: {'order_id': '34', 'product_id': '44827', 'quantity': 51, 'type': 'delivery', 'timestamp': '4/18/2024'}
Received delivery task: {'order_id': '766', 'product_id': '69', 'quantity': 86, 'type': 'delivery', 'timestamp': '6/16/2023'}
Received delivery task: {'order_id': '27116', 'product_id': '91', 'quantity': 75, 'type': 'delivery', 'timestamp': '1/19/2024'}
Received delivery task: {'order_id': '29', 'product_id': '2', 'quantity': 88, 'type': 'delivery', 'timestamp': '3/12/2024'}
```

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