# Kafka Integration in Audit Manager Application

## 1. Introduction

In the Audit Manager application, Apache Kafka is used to efficiently manage and log audit data. The process involves retrieving data from the source database (Excel file), processing it, and sending it to a Kafka topic. This data is then consumed by a Kafka consumer, which updates the audit log database. The entire workflow is initiated through a batch file that starts both the Kafka producer and consumer applications, followed by an API-triggered data fetch.

## 2. Architecture Overview

The architecture includes:  
- Kafka Producer: Fetches data from the source database(Excel file) via an API call and sends it to a Kafka topic.

**(Location of Excel file : C:\Audit\_Report\CadData\_Sample\_Data\_audit\_object\_change\_tracker\_Test\_updated1.xlsx)**  
- Kafka Topic: Acts as a message broker that stores messages containing data for further processing.  
- Kafka Consumer: Retrieves messages from the Kafka topic and updates the audit log database.

Data Flow Overview:  
1. Batch File Execution: The batch file starts both producer and consumer applications.  
2. API Data Fetch and Publish: The API fetches required data from the source database(Excel file) and pushes it to the Kafka topic via the Kafka producer.  
3. Consumer Processing and Audit Log Update: The Kafka consumer reads messages from the topic and processes them to update the audit log database.

## 3. Project Structure

### Audit Consumer Structure:

Audit-Consumer/  
├── src/  
│ ├── main/  
│ │ ├── java/com/acnovate/auditmanager/  
│ │ │ ├── config/  
│ │ │ │ ├── security/RestAPIConfigurator.java  
│ │ │ │ ├── KafkaOffsetConfiguration.java  
│ │ │ │ └── RestTemplateConfiguration.java  
│ │ │ ├── dao/  
│ │ │ │ ├── AuditAttributeChangeTrackerRepository.java  
│ │ │ │ ├── AuditObjectChangeTrackerRepository.java  
│ │ │ │ └── SourceReferenceObjectRepository.java  
│ │ │ ├── domain/  
│ │ │ │ ├── dto/  
│ │ │ │ │ ├── event/  
│ │ │ │ │ │ ├── AuditEvent.java  
│ │ │ │ │ │ ├── AuditEventMetadata.java  
│ │ │ │ │ │ └── AuditEventPayload.java  
│ │ │ │ └── entity/  
│ │ │ │ ├── AuditAttributeChangeTracker.java  
│ │ │ │ ├── AuditEntity.java  
│ │ │ │ ├── AuditObjectChangeTracker.java  
│ │ │ │ └── SourceReferenceObject.java  
│ │ │ ├── errortemplate/  
│ │ │ │ └── ErrorTemplate.java  
│ │ │ ├── exception/  
│ │ │ │ ├── MaybeRecoverableException.java  
│ │ │ │ ├── MessageDeserializationException.java  
│ │ │ │ ├── MessageProcessingTimeoutException.java  
│ │ │ │ ├── RecoverableException.java  
│ │ │ │ └── UnsupportedEventException.java  
│ │ │ ├── handler/  
│ │ │ │ ├── AuditAuditReportHandler.java  
│ │ │ │ └── MessageHandler.java  
│ │ │ ├── listener/  
│ │ │ │ ├── AbstractBaseKafkaListener.java  
│ │ │ │ ├── AuditEtlListener.java  
│ │ │ │ └── KafkaSeekOffsetAwareListener.java  
│ │ │ ├── service/  
│ │ │ │ └── UnmarshallerService.java  
│ │ │ └── AuditConsumerApplication.java  
│ └── resources/  
│ ├── application.properties  
│ └── application.yml

### Audit Producer Structure:

Audit-Manager/  
├── src/  
│ ├── main/  
│ │ ├── java/com/acnovate/auditmanager/  
│ │ │ ├── controller/  
│ │ │ │ └── AuditReportController.java  
│ │ │ ├── dto/  
│ │ │ │ ├── AuditEvent.java  
│ │ │ │ ├── AuditEventMetadata.java  
│ │ │ │ ├── AuditEventPayload.java  
│ │ │ │ ├── AuditRequest.java  
│ │ │ │ └── AuditResponse.java  
│ │ │ ├── service/  
│ │ │ │ ├── AuditReportService.java  
│ │ │ │ └── FileReader.java  
│ │ │ ├── AuditManagerApplication.java  
│ │ │ └── KafkaProducer.java  
│ └── resources/  
│ ├── application.properties  
│ └── application.yml  
└── test/  
 ├── java/com/acnovate/auditmanager/  
 │ └── AuditManagerApplicationTests.java

## 4. Starting Kafka server

Go to your Kafka server folder location and move to windows batch file location:

**Ex: C:\kafka\kafka\_2.12-3.6.1\bin\windows**

1. Open command prompt and start Zookeeper by running following command.

**zookeeper-server-start.bat ..\..\config\zookeeper.properties**

1. Open command prompt and start server by running following command.

**kafka-server-start.bat ..\..\config\server.properties**

1. Open command prompt and create topic by running following command.

**kafka-topics.bat --create --topic AuditManager --bootstrap-server localhost:9092 --replication-factor 1 --partitions 2**

1. Goto following location and run the Batch file to run producer and consumer jars.

**C:\Audit\_Report**

**Run : start\_apps.bat**

## 5. Running Kafka Producer and Consumer with Batch Job

To initiate the Kafka-based data flow, batch jobs have been set up to run the JAR files for each application.

### Execution Process:

1. Creating JAR Files: Each application (Audit Producer and Audit Consumer) is built into its own JAR file.  
2. Batch File Execution: A batch file (runKafkaJobs.bat) is configured to start both JAR files for producer and consumer applications.

Sample runKafkaJobs.bat content:

@echo off  
echo Starting Kafka Producer...  
java -jar path o\AuditProducer.jar  
  
echo Starting Kafka Consumer...  
java -jar path o\AuditConsumer.jar  
  
echo Kafka Producer and Consumer have started.  
pause

3. Execution: Running this batch file once initiates the Kafka producer and consumer applications. This workflow enables efficient data retrieval, processing, and audit log updates.

## 6. Use Case Example: Audit Log Update

### Step-by-Step Workflow:

1. Batch File Trigger: The batch file is executed to start the producer and consumer applications.  
2. API Data Fetching: The producer fetches data from the source database(Excel file) through an API call.  
3. Kafka Producer: Sends the fetched data as messages to the Kafka topic.  
4. Kafka Topic: Holds the messages for the consumer to process.  
5. Kafka Consumer: Reads messages from the topic and updates the audit log database.

### Example Scenario:

If there is a database change at 10:00 AM, the Kafka producer sends this data to the Kafka topic after being triggered, and the Kafka consumer processes the message to update the audit log with the new information.

## 7. Error Handling and Retries

- Retries: Configured retries in case of failed message deliveries.  
- Dead Letter Topics: Messages that repeatedly fail are sent to a dead letter topic.  
- Backoff Strategy: Consumers retry processing failed messages with a delay to avoid overload.

## 8. Monitoring and Maintenance

To ensure Kafka is running efficiently, the following metrics are monitored:  
- Lag Monitoring: Checks that the consumers are processing messages as expected.  
- Throughput: Measures message processing rates to identify bottlenecks.  
- Consumer Lag: Tracks how far behind the consumer is from the latest produced messages.

## Conclusion

Kafka provides a robust and efficient solution for managing data flow in the Audit Manager application. By running the producer and consumer applications through a batch job and handling data retrieval via API, the system maintains a reliable log of audit data. This setup ensures timely and accurate audit log updates, with comprehensive monitoring for continued performance.