Recommendation System + Kafka Integration Documentation

# Overview

This document details the steps taken to successfully integrate Apache Kafka into a Flask-based hybrid recommendation system API. The system supports collaborative filtering, content-based filtering, and fallback mechanisms, enhanced with MMR diversification. Kafka is used to log recommendation events in real-time.

# System Architecture

The system includes the following components:

* - Flask API (`api.py`) to serve recommendations.
* - MongoDB as the backend database for users, reviews, restaurants, etc.
* - Kafka running in Docker using Bitnami images.
* - A Kafka producer in Flask to send recommendation logs.
* - A Kafka consumer script to monitor messages in real-time.

# Kafka Setup via Docker

Kafka and Zookeeper were launched using Bitnami images. Final working Docker command:

docker run -d --name kafka --link zookeeper:zookeeper -e KAFKA\_CFG\_KRAFT\_MODE=false   
-e KAFKA\_CFG\_PROCESS\_ROLES=broker -e KAFKA\_CFG\_NODE\_ID=1   
-e KAFKA\_CFG\_ZOOKEEPER\_CONNECT=zookeeper:2181 -e KAFKA\_CFG\_LISTENERS=PLAINTEXT://:9092   
-e KAFKA\_CFG\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:9092   
-e KAFKA\_CFG\_LISTENER\_SECURITY\_PROTOCOL\_MAP=PLAINTEXT:PLAINTEXT -e KAFKA\_BROKER\_ID=1   
-e ALLOW\_PLAINTEXT\_LISTENER=yes -p 9092:9092 bitnami/kafka:3.4.0-debian-11-r17

# Flask Integration

The following files were created/modified:

* - `kafka\_producer.py`: Kafka producer wrapper
* - `api.py`: Modified to import and send messages via Kafka after recommendations
* - `kafka\_consumer.py`: Optional real-time consumer to verify messages

# Example Kafka Message Structure

{  
 "user\_id": "6821cb8fb5ce7f4dd4d2abca",  
 "type": "product",  
 "result": {  
 "Recommendations": [...],  
 "Products": {...},  
 "Neighbor Preferences": {...},  
 "Target User Preference": {...},  
 ...  
 }  
}

# Validation and Testing

After sending a request to the Flask endpoint, the Kafka consumer displayed the expected message, verifying that the end-to-end flow was functional.

# Suggested Next Steps

* - Create a feedback endpoint to capture LIKE/DISLIKE actions.
* - Log Kafka messages to MongoDB or flat files for analytics.
* - Trigger retraining or model updates based on streaming data.