

Real-Time Flight Status Monitoring Pipeline Using Apache Kafka

1. Business Use Case: Real-Time Flight Tracker for Travel Agencies

Objective:

To provide a **real-time flight status monitoring system** for travel agencies using **Apache Kafka**, enabling them to track delays, cancellations, and changes in schedules. This improves customer service, resource planning, and operational efficiency.

Justification:

In the aviation and travel industry, even small delays or cancellations can cause massive ripple effects, impacting customer satisfaction and operational cost. Agencies require an automated and scalable system that can:

- Detect flight status changes immediately.
- Alert internal teams and customers.
- Maintain a historical database of disruptions.

This real-time data capture pipeline solves those needs using reliable tools like **Kafka** and **MongoDB**, offering both low-latency processing and long-term storage.

API Used:

- **AviationStack API:** A free, RESTful API providing real-time flight information worldwide.
- **Data Format:** JSON objects including fields such as flight number, airline, departure/arrival airports and times, and flight status.

2. Architecture Overview

Components Involved:

1. Data Source (AviationStack API):

- Acts as the external data provider.
- Flight data is fetched via HTTP requests in real time.

2. Kafka Producer (Python):

- Fetches data at fixed intervals (60 seconds).
- Parses relevant fields and sends JSON messages to Kafka.

3. Kafka Broker:

- Receives data from the producer.
- Writes to the `flight-topic` topic.
- Ensures delivery guarantees and message persistence.

4. Kafka Consumer (Python):

- Subscribes to `flight-topic`.
- Reads incoming messages in near real-time.
- Sends messages for processing or storage.

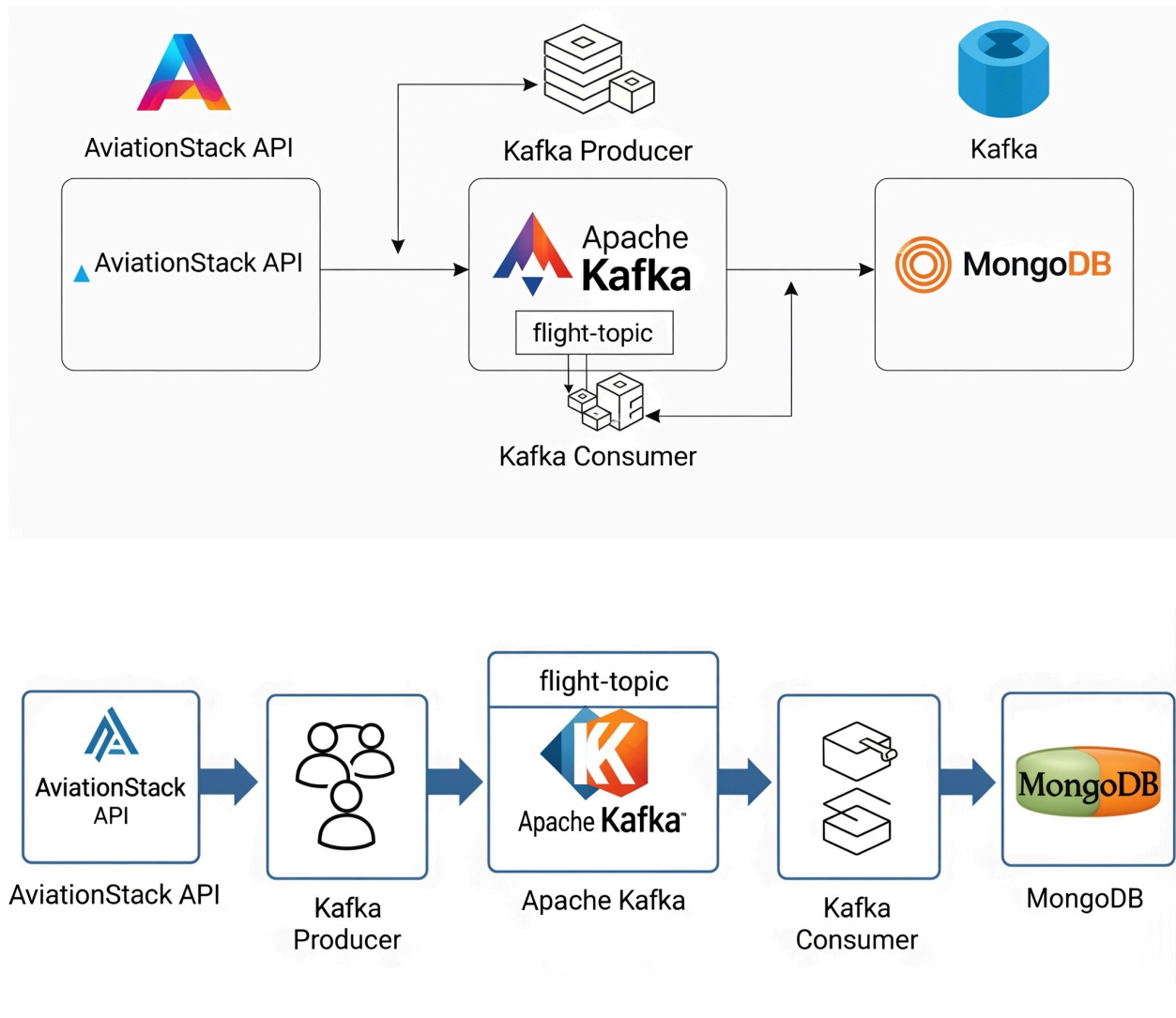
5. MongoDB (NoSQL Database):

- Stores all flight records for historical access, analytics, and future dashboarding.

6. Optional Alert Layer or Dashboard:

- Triggers based on flight status (e.g., "delayed").
- Can be built using Python, Streamlit, Node.js, or Power BI.

Diagram:



3. How the Pipeline Captures, Processes, and Delivers Real-Time Data

1. Data Capture:

- **Source:** AviationStack API is queried at regular intervals (every 60 seconds).
- **Content:** JSON data for multiple flights, including departure and arrival details, current status, and timestamps.

- **Kafka Producer:** Parses and serializes the relevant fields from API into structured JSON messages.

2. Data Processing:

- Kafka acts as a high-throughput, fault-tolerant stream processing system.
- Messages are written to the `flight-topic`.
- The Kafka Consumer reads these messages in real-time.
- Intermediate validations, enrichments, or filtering logic can be implemented here (e.g., flagging only delayed flights).

3. Data Delivery:

- The processed messages are delivered to MongoDB using `pymongo`.
- Each message becomes a new document in the database.
- This delivery enables historical logging, querying, reporting, and alert triggers.
- Other downstream systems can subscribe to this data for real-time dashboards or alert systems.

4. Code Overview & Technologies Used

Producer:

- **Python** with `requests` and `kafka-python`.
- Scheduled with a `while` loop and `time.sleep()`.
- Handles API errors, empty responses, and retries.

Consumer:

- Uses `kafka-python` and `pymongo`.
- Listens continuously for new Kafka messages.
- Inserts JSON documents into MongoDB.

MongoDB:

- Acts as a real-time data sink.
- Ideal for time-series flight data.
- Supports queries for aggregation, analysis, and visualization.

5. Benefits and Justifications

Feature	Benefit
Real-Time Streaming	Instant updates for flight changes
Scalable	Kafka can handle high data throughput and multiple consumers
Reliable	Messages are stored durably until processed
Flexible	MongoDB allows dynamic data models for unstructured data
Extensible	Easily integrates with dashboards, alert engines, and machine learning models

This architecture ensures **low-latency**, **high-throughput**, and **flexible processing**, which is essential in industries where timing is critical (e.g., aviation).

6. Future Enhancements

- **Alerting Engine:** Send alerts via email/SMS for flight disruptions.
- **Dashboard Integration:** Use Streamlit, Power BI, or Grafana to build live visualizations.
- **Machine Learning Layer:** Predict delays based on historical patterns.
- **Data Partitioning:** Scale Kafka by partitioning topics by airport or airline.

7. Conclusion

This real-time data capture pipeline using Apache Kafka, Python, and MongoDB provides a robust foundation for monitoring flight statuses in real time. It is suitable for travel agencies, airports, and logistics firms to improve customer communication, optimize resource deployment, and build proactive services.

The architecture is flexible, scalable, and aligned with modern event-driven system design — making it both technically sound and business-relevant for real-world deployment.

8. Producer Code

```
import time
import json
import requests
from kafka import KafkaProducer, errors

# --- Configuration ---
API_KEY = '47fd3566d9e2432fed163527db935eb0' # ✅ Your API key
KAFKA_BROKER = 'localhost:9092'
TOPIC = 'flight-topic'
FETCH_INTERVAL = 60 # Free plan allows 1 call/min

# --- Kafka Producer Setup ---
print("Connecting to Kafka broker...")
try:
    producer = KafkaProducer(
```

```

        bootstrap_servers=KAFKA_BROKER,
        value_serializer=lambda v: json.dumps(v).encode('utf-8'),
        retries=5,
        acks='all'
    )

    producer.partitions_for(TOPIC)
    print(f"✅ Connected to Kafka broker and topic '{TOPIC}'")
except errors.NoBrokersAvailable:
    print("❌ Kafka broker unavailable. Make sure Kafka is running.")
    exit(1)

# --- Build AviationStack API URL ---
def build_url(api_key):
    return
    f"http://api.aviationstack.com/v1/flights?access_key={api_key}&limit=5"

# --- Fetch and Stream Flight Data ---
print(f"\n📡 Starting flight status stream every {FETCH_INTERVAL}
seconds...")

try:
    while True:
        try:
            url = build_url(API_KEY)
            response = requests.get(url)
            data = response.json()

            if 'data' not in data:
                print("⚠️ No flight data returned. Waiting for next
attempt...")

            print("🔴 Raw response:", data)
            time.sleep(FETCH_INTERVAL)
            continue

            flights = data['data']

            for flight in flights:
                record = {
                    'airline': flight.get('airline', {}).get('name'),
                    'flight_number': flight.get('flight', {}).get('iata'),

```

```

        'departure_airport': flight.get('departure',
{}).get('airport'),
        'departure_scheduled': flight.get('departure',
{}).get('scheduled'),
        'arrival_airport': flight.get('arrival',
{}).get('airport'),
        'arrival_scheduled': flight.get('arrival',
{}).get('scheduled'),
        'status': flight.get('flight_status'),
        'timestamp': time.time()
    }

    producer.send(TOPIC, value=record)
    producer.flush()
    print(f"✅ Produced: {record}")

except Exception as e:
    print(f"❌ Error fetching or sending data: {e}")

time.sleep(FETCH_INTERVAL)

except KeyboardInterrupt:
    print("\n🛑 Stream stopped by user.")
finally:
    producer.close()
    print(f"✅ Kafka producer closed.")

```

9. Consumer and MongoDB Code

```

import json
from kafka import KafkaConsumer
from pymongo import MongoClient

# --- Configuration ---
KAFKA_BROKER = 'localhost:9092'
TOPIC = 'flight-topic'
GROUP_ID = 'flight-consumer-group'

```



```

# MongoDB setup
MONGO_URI =
'mongodb+srv://karanmakoll:DaenerysDrag@cluster0.zjqxxsb.mongodb.net/'
DB_NAME = 'flight_data'
COLLECTION_NAME = 'flight_status'

# --- Connect to MongoDB ---
try:
    mongo_client = MongoClient(MONGO_URI)
    db = mongo_client[DB_NAME]
    collection = db[COLLECTION_NAME]
    print(f"✅ Connected to MongoDB. DB: '{DB_NAME}', Collection:
'{COLLECTION_NAME}'")
except Exception as e:
    print(f"❌ Failed to connect to MongoDB: {e}")
    exit(1)

# --- Kafka Consumer Setup ---
print(f"🔄 Connecting to Kafka topic '{TOPIC}'...")
consumer = KafkaConsumer(
    TOPIC,
    bootstrap_servers=KAFKA_BROKER,
    group_id=GROUP_ID,
    auto_offset_reset='latest',
    enable_auto_commit=True,
    value_deserializer=lambda v: json.loads(v.decode('utf-8'))
)

print(f"✅ Listening for flight data on topic '{TOPIC}'...\n")

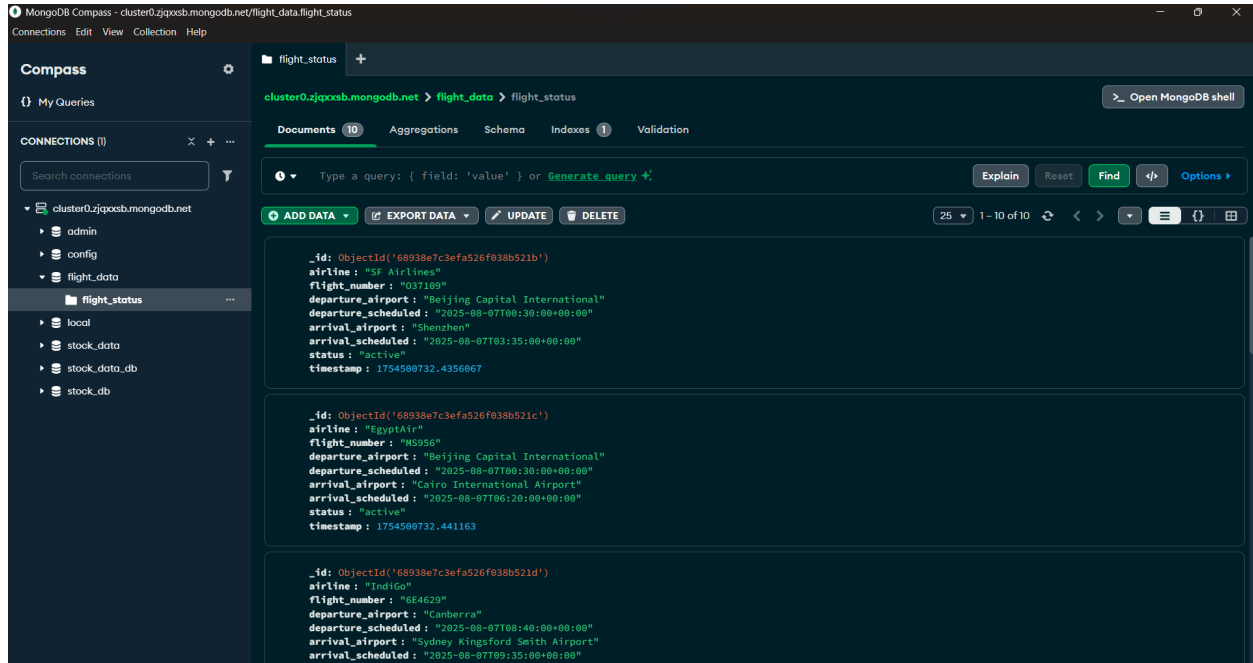
try:
    for message in consumer:
        record = message.value
        print(f"✈️ Inserting Flight Record: {record}")
        collection.insert_one(record)
except KeyboardInterrupt:
    print("\n🛑 Consumer stopped by user.")
finally:
    consumer.close()

```

```
mongo_client.close()

print("✅ Kafka consumer and MongoDB connection closed.")
```

9. Consumed Message into MongoDB Compass



The screenshot shows the MongoDB Compass interface. The left sidebar displays the 'CONNECTIONS (1)' section with a search bar and a tree view of the database structure. The main panel shows the 'flight_status' collection with 10 documents. The 'Documents' tab is selected, and the query bar is empty. The document list shows three documents with the following fields: _id, airline, flight_number, departure_airport, departure_scheduled, arrival_airport, arrival_scheduled, status, and timestamp.

_id	airline	flight_number	departure_airport	departure_scheduled	arrival_airport	arrival_scheduled	status	timestamp
ObjectId('68938e7c3efa526f838b521b')	SF Airlines	037109	Beijing Capital International	2025-08-07T00:30:00+00:00	Shenzhen	2025-08-07T03:35:00+00:00	active	1754500732.4356067
ObjectId('68938e7c3efa526f838b521c')	EgyptAir	MS956	Beijing Capital International	2025-08-07T00:30:00+00:00	Cairo International Airport	2025-08-07T06:20:00+00:00	active	1754500732.441163
ObjectId('68938e7c3efa526f838b521d')	IndiGo	6E4629	Canberra	2025-08-07T00:40:00+00:00	Sydney Kingsford Smith Airport	2025-08-07T09:35:00+00:00		