# Real-Time Alert Analysis System

## Problem Statement

Currently, the system relies on manually downloaded CSV files containing logs from Kibana (Elasticsearch backend) to perform alert analysis.   
This approach is not scalable for the following reasons:  
1. Manual Effort: Logs must be manually exported from Kibana, introducing delays and inefficiencies.  
2. Data Volume Growth: As the number of alerts increases, handling large CSV files will slow down the data ingestion, processing, and visualization processes.  
3. Real-Time Insights: The system cannot process logs in real-time, which is critical for proactive monitoring and alerting.  
4. Scalability: The current setup is not scalable to handle the expected growth in log data.

## Proposed Solution

To address these challenges, we will build a real-time alert analysis pipeline leveraging:  
- Apache Kafka for real-time data streaming.  
- Elasticsearch APIs to programmatically fetch logs from the Kibana backend.  
- A real-time database (e.g., PostgreSQL, TimescaleDB) to store and query processed data.  
- Streamlit for visualizing real-time insights from the database.

## System Architecture

### Workflow

1. Data Source (Elasticsearch/Kibana):  
 - Logs are retrieved from Elasticsearch using REST APIs.  
 - Logs include fields such as @timestamp, EventSource, EventLevel, and EventTitle.  
  
2. Data Streaming (Apache Kafka):  
 - Kafka Producer sends logs fetched from Elasticsearch into a Kafka topic (alerts\_stream).  
 - Kafka Topic acts as a distributed buffer, storing the logs.  
  
3. Data Processing (Kafka Consumer):  
 - Kafka Consumer reads logs from the Kafka topic and processes them.  
 - Processed logs are stored in a real-time database for analysis.  
  
4. Real-Time Visualization (Streamlit):  
 - Streamlit fetches processed data from the database to generate interactive visualizations and dashboards.  
 - Visualizations include daily trends, hourly breakdowns, heatmaps, and predictive analytics.

## Implementation Plan

### 1. Data Source: Kibana/Elasticsearch

Logs will be fetched from Elasticsearch using its REST API. This eliminates manual CSV downloads and allows for near real-time data ingestion.  
  
Steps:  
1. Use Elasticsearch’s \_search API to fetch logs within a specific time range.  
 Example query to fetch logs from the last 15 minutes:  
 GET /\_search  
 {  
 "query": {  
 "range": {  
 "@timestamp": {  
 "gte": "now-15m"  
 }  
 }  
 }  
 }  
2. Write a Python script to execute this query and parse the response.

### 2. Kafka Setup

Kafka will handle real-time data streaming between Elasticsearch (producer) and the consumer.  
  
Steps:  
1. Install Kafka and Zookeeper:  
 Follow the Kafka Quickstart Guide.  
2. Create a Kafka Topic:  
 bin/kafka-topics.sh --create --topic alerts\_stream --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1  
3. Kafka Producer:  
 Write a Python script to fetch logs from Elasticsearch and send them to Kafka.

### 3. Kafka Consumer

The Kafka consumer reads logs from the alerts\_stream topic and stores them in a real-time database.  
  
Steps:  
1. Set Up a Database:  
 Install PostgreSQL or TimescaleDB for storing processed data.  
 Create a table to store alerts.  
2. Kafka Consumer:  
 Write a Python script to read logs from Kafka and insert them into the database.

### 4. Real-Time Database

Use PostgreSQL or TimescaleDB to store alerts. Aggregate data for better performance:  
- Hourly counts  
- Daily trends

### 5. Streamlit Visualization

Streamlit will query the database for visualizations.  
  
Steps:  
1. Replace CSV with database queries.  
2. Create real-time visualizations:  
 - Bar charts for daily trends.  
 - Heatmaps for hourly alerts.  
 - Predictive analytics (e.g., ARIMA model) for future trends.

## Future Scalability

1. High-Performance Databases:  
 - Use ClickHouse for large-scale analytics.  
 - Use Elasticsearch for querying structured and unstructured logs.  
  
2. Distributed Processing:  
 - Use Apache Spark or Flink to process Kafka streams at scale.  
  
3. Real-Time Alerting:  
 - Add real-time alerting capabilities (e.g., email, Slack notifications) for critical events.  
  
4. Dashboard Improvements:  
 - Build advanced dashboards using tools like Grafana or Kibana alongside Streamlit.

## Conclusion

This architecture leverages Kafka for real-time streaming, Elasticsearch for log ingestion, and Streamlit for visualization,   
creating a scalable and automated alert analysis system. It addresses current limitations and ensures the system can handle   
increasing data volumes efficiently.