---

# Algorithm: Real-time Log Analysis Using Hadoop and Spark

## Step 1: Data Collection

- **1.1 Source Identification**
  - Identify log sources such as web servers (e.g., Apache, Nginx), applications, and system logs.
- **1.2 Log Shippers**
  - Configure Apache Flume or Apache Kafka agents on each server to collect and forward logs.
- **1.3 Streaming Ingestion**
  - Logs are continuously published to Kafka topics or Flume channels.

## Step 2: Data Ingestion into Spark

- **2.1 Stream Setup**
  - Set up an Apache Spark Structured Streaming job to subscribe to the Kafka topic.
- **2.2 Batching**
  - Spark reads log messages in micro-batches (e.g., every 5 or 10 seconds).
- **2.3 Deserialization & Parsing**
  - Each log message is parsed into structured fields (e.g., timestamp, IP, URL, status code) using regular expressions or log parsers.

## Step 3: Data Processing & Transformation

- **3.1 Filtering**
  - Filter logs for relevant patterns, such as ERROR messages or HTTP status codes ≥ 500.
  - Optionally, discard health checks and known benign events.
- **3.2 Transformations**
  - Extract required fields (map step).
  - Normalize and enrich the data as needed (e.g., GeoIP lookup for IP addresses, user-agent parsing).
- **3.3 Windowed Aggregation**
  - Use Spark's window functions to aggregate logs over sliding or tumbling windows (e.g., count errors per 10 minutes with a 2-minute slide).
- **3.4 Aggregation Functions**
  - Compute metrics such as:
    - Hit counts per URL, IP, and status code.

        - Error rates over time.
        - Average response size or latency (if available).

### Step 4: Analysis & Anomaly Detection

- **4.1 Trend Analysis**
    - Track high-frequency sources (e.g., the IP with the most requests).
    - Identify top-accessed URLs and their response status distribution.
- **4.2 Anomaly Detection**
    - Monitor for sudden spikes in error rates by comparing against moving averages.
    - Optionally, apply MLlib for clustering or classification to perform unsupervised detection of unusual activity (e.g., DDoS attacks).
- **4.3 Alerting**
    - If predefined thresholds are breached (e.g., error rate > 5% in any window), trigger alerts via dashboards, email, or other notification systems.

### Step 5: Data Persistence

- **5.1 Storing to HDFS/Hive**
    - Store all raw and processed logs to HDFS in partitioned folders.
- **5.2 Batch Table Updates**
    - Update or append to Hive tables for historical analytics, partitioned by date and hour.
- **5.3 Exporting to Search Engines**
    - (Optional) Index processed summaries in Elasticsearch for near-real-time search and exploration.

### Step 6: Visualization & Reporting

- **6.1 Dashboards**
    - Visualize live metrics (top URLs, error rates, traffic spikes) in tools like Kibana, Grafana, or Power BI by querying your data sinks (Hive, Elasticsearch, etc.).
- **6.2 Trend Reports**
    - Generate reports (e.g., hourly or daily summaries, anomaly logs) for historical review.
- **6.3 Alerts**
    - Display or send alerts on current and historical anomalies through dashboards or push notifications.

---

## Inputs and Outputs

- **Inputs:**
    - Raw logs containing fields such as timestamp, IP, URL, method, status, size, user-agent, referrer, etc.

    - Data sourced from Kafka/Flume or direct file tailing.
- **Outputs:**
    - Aggregated metrics (per minute/hour/day).
    - Lists of top URLs and IPs.
    - Anomaly and alert logs with context.
    - Interactive dashboards and exported reports.
    - Raw and analyzed logs stored for historical queries.

---

# Pseudocode Summary

```
For each incoming log from Kafka (every batch_interval seconds)
  Parse log line into fields (timestamp, ip, url, method, status, etc.)
  If status >= 500 or matches ERROR filter
    Increment error counter (by URL, IP, time window)
  Increment total request counter (by URL, IP, time window)
  Store parsed record in temporary DataFrame

  For each aggregation window (e.g., 10 min sliding)
    Calculate:
      - Total requests per IP, per URL, per status
      - Error rate per window
    If error rate > threshold OR anomaly detected:
      Record alert event

  Save batch results:
    - Store detailed logs to HDFS (partitioned by time)
    - Store window summary to Hive/SparkSQL tables

  Export current aggregates to Elasticsearch (for live dashboard) if enabled

Dashboard/Reports:
  Query stored aggregates for visualization (real-time and historical)
  Show top URLs, error spikes, traffic patterns, and raise alerts if needed
```

Salendram Sowmya Sri
24M11MC153
Aditya Univeristy