Apache Flume

- It is a distributed, reliable, and scalable service for efficiently collecting, aggregating, and moving large amounts of log data from many different sources to a centralized data store, typically Hadoop's HDFS.
- It is designed to be highly extensible, allowing it to integrate with a variety of sources, sinks, and processing mechanisms.
- It is often used in big data applications where real-time data ingestion from various sources like web servers, application logs, social media feeds, and more needs to be efficiently processed and stored.

Features of Apache Flume

- **Distributed:** Flume can operate in a distributed, fault-tolerant, and scalable manner, making it suitable for large data volumes.
- **Reliable:** It provides reliability and guarantees data delivery to the specified sinks, even in the case of node failures.
- **Flexible Data Sources:** Flume supports a wide variety of data sources, including file systems, HTTP endpoints, custom applications, and more.
- **Multiple Sinks:** Flume can push data to various sinks like HDFS, HBase, Kafka, and other data storage systems.
- **Data Transformation:** Flume allows for basic data processing and transformation through interceptors and other mechanisms before sending data to sinks.

Flume Components

Source:

- The source receives data from an external entity and pushes it into Flume.
- Common sources include log files, HTTP endpoints, and custom sources.
- Examples include ExecSource (reads from a shell command), SpoolDirectorySource (reads from directories), and AvroSource (reads from Avro).

Channel:

- Channels are intermediate buffers between sources and sinks.
- Data flows through the channel before being forwarded to the sink.
- There are two main types of channels:
    - Memory Channel (in-memory buffer, faster but less reliable).
    - File Channel (on-disk buffer, more reliable but slower).

Flume Components

Sink:

- Sink is responsible for delivering data to a destination.
- Popular sinks include HDFS, HBase, Kafka, File, and ElasticSearch

Interceptor (Optional):

- Interceptors provide a way to modify or filter events between the source and the channel. They can be used to filter, transform, or drop events.

Agent:

- Flume agent is the entity that runs the source, channel, and sink configuration.
- An agent can be run on a single machine or across multiple machines for scaling.

Configuration File:

- Flume is configured using a configuration file, usually in a simple key-value format, which defines the sources, channels, and sinks.

Flume Working

Data Ingestion: Flume collects data from various sources (e.g., log files, databases, and network streams).

Data Flow: The data flows through channels, where it may be buffered temporarily before being sent to the appropriate sink.

Data Delivery: The data is delivered to a destination sink, such as HDFS, for storage or further processing.

Use of Apache Flume - Examples

Log Aggregation: Collecting log data from various applications, web servers, and systems and storing it in a centralized location like HDFS for further analysis.

Streaming Data Ingestion: Ingesting real-time data from various streaming sources into big data systems like HDFS or HBase for processing and analysis.

Data Movement: Moving data between systems in a distributed and scalable manner, ensuring reliability in data delivery.

Real-Time Data Pipelines: Using Flume as a part of a real-time data pipeline where logs and other streaming data are ingested, processed, and stored.

Flume vs Kafka

Both are used for streaming data ingestion

Flume is optimized for log data aggregation and delivery to systems like HDFS.

Kafka is designed as a high-throughput distributed messaging system for real-time stream processing and can be used for various types of data streams beyond logs.

In short:

Flume is more suited for ingesting logs and sending data to big data systems (HDFS, HBase, etc.).

- **Kafka** is more suited for managing real-time data streams and offering publish/subscribe messaging with distributed stream processing.

Real world Example- Apache Flume

- **Apache Flume** being used to collect, aggregate, and move log data from web servers to **HDFS** for further analysis.
- **Scenario: Consider a simple example of Flume configuration that reads log data from a file source, uses a memory channel, and writes it to HDFS.**
- That is, simulate the ingestion of web server log files and store them into HDFS, where they can be analyzed for patterns or errors.
- Flume uses a configuration file to define the source, channel, and sink.
    - **Source**: Web server logs (e.g., Apache or Nginx logs).
    - **Sink**: HDFS for storage and later analysis.
    - **Channel**: In-memory channel for faster throughput.
    - **Interceptor**: Optional filtering or transformation of the logs before being sent to HDFS.

- **Source (webSource)**:
    - This *uses the exec source*, which runs an external command (*tail -F /var/log/apache2/access.log*) to read new log entries from a web server log file in real-time.
    - The -F option makes tail follow the log file indefinitely, outputting new entries as they arrive.
    - Source sends the log data into the **memory channel** for buffering before it is written to HDFS.
- **Channel (memoryChannel):**
    - The memory channel temporarily holds data in memory. It's fast but may be less reliable than file-based channels because if the agent crashes, in-memory data could be lost.
    - The capacity parameter sets the maximum number of events that can be stored in the memory channel before they are flushed to the sink (HDFS).
    - *transactionCapacity* determines how many events are grouped into a transaction before being written to HDFS.
- **Sink (hdfsSink):**
    - The **HDFS sink** writes the collected log data into **HDFS**.
    - The logs are stored under the path */flume/logs/* on HDFS.
    - *rollSize* and *rollCount* define when to create a new file. Once a file reaches 128MB (rollSize) or 10,000 events (rollCount), a new file is created.

How Works

Step 1: Flume's *webSource* starts tailing the log file ***(access.log***) and sends new log lines into the **memory channel**.

Step 2: **Memory channel** buffers the incoming log lines temporarily.

Step 3: Once the buffer reaches the specified *transactionCapacity* (100 events in this case), it pushes the events to the **HDFS sink**.

Step 4: HDFS sink writes the log data to HDFS under */flume/logs/,* creating new log files when the roll size or roll count is reached.

```
# Define the agent name and components used
agent.sources = webSource
agent.channels = memoryChannel
agent.sinks = hdfsSink

# --- Source Configuration ---

# Using an Exec source to tail a log file (simulating an Apache web server log)
webSource.type = exec

# The command `tail -F /var/log/apache2/access.log` will stream new lines as they are
added to the log file.
webSource.command = tail -F /var/log/apache2/access.log
# Set the number of threads to run the command (e.g., 1 thread)
webSource.channels = memoryChannel

# --- Channel Configuration ---

# Use memory channel to store events temporarily before they are written to the sink.
memoryChannel.type = memory
# Define the channel capacity, which dictates how many events can be stored in memory
at a time
memoryChannel.capacity = 1000
# Define the transaction capacity, which is how many events will be written to the sink
in one batch
memoryChannel.transactionCapacity = 100
```

```
# --- Sink Configuration ---

# Use HDFS sink to write the log data into HDFS
hdfsSink.type = hdfs
# Specify the HDFS path where the logs will be stored
hdfsSink.hdfs.path = hdfs://namenode/flume/logs/
# Prefix for files that will be created in HDFS
hdfsSink.hdfs.filePrefix = apache_logs_
# Specify the maximum file size (rolls over when reached)
hdfsSink.hdfs.rollSize = 134217728
# Specify the number of events to accumulate before rolling over to a new file
hdfsSink.hdfs.rollCount = 10000

# Specify the type of data storage format (using Avro here for structured log events)
hdfsSink.hdfs.writeFormat = Text
•  hdfsSink.channel = memoryChannel
```

Running Flume Agent

Once the configuration file is ready, run Flume agent using the following command:

flume-ng agent --conf ./conf --conf-file flume-conf.properties --name agent -Dflume.root.logger=INFO,console

where

- *--conf ./conf*: Specifies the directory where Flume's configuration files are stored.
- *--conf-file flume-conf.properties*: Points to created Flume configuration file.
- *--name agent*: Specifies the name of the Flume agent to run (the agent's components are referenced by this name).
- *-Dflume.root.logger=INFO,console*: This logs the output of Flume to the console for debugging purposes.

Viewing Data in HDFS

You can check the data stored in HDFS with the following command:

hadoop fs -ls /flume/logs/

You should see files named something like *apache_logs_000000000000_0001* in the specified HDFS directory.

where,

apache_logs: The prefix or base name of the file, usually specified in the Flume configuration or output format of a MapReduce job.

000000000000: A task or file sequence number, typically auto-incremented starting from zero.

0001: A block index or replica identifier — in some contexts (especially with MapReduce), this may be part of the task number or split output.

E-commerce Clickstream and Transaction Analysis

Clickstream: Sequence of clicks or actions a user performs while navigating a website or application like a digital breadcrumb trail.

A breadcrumb captures:

- Which pages a user visits or screen
- In what order they were accessed
- For how long, often includes timestamped user actions
- What items they click on (links, buttons, products, ads, etc.)

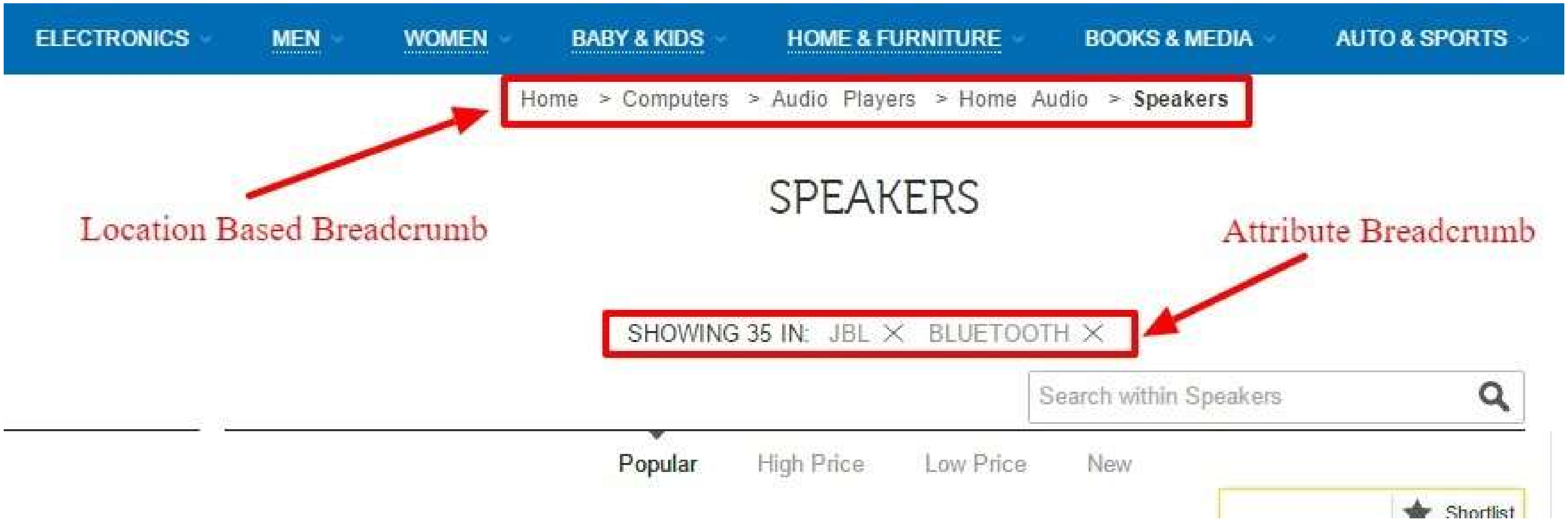
Breadcrumb trail helps analysts understand:

- How users move through a site (navigation flow)
- Where they drop off or convert
- Which paths are most common or problematic

Example: A user's breadcrumb trail is:

Homepage → Search → Product Detail → Cart → Checkout

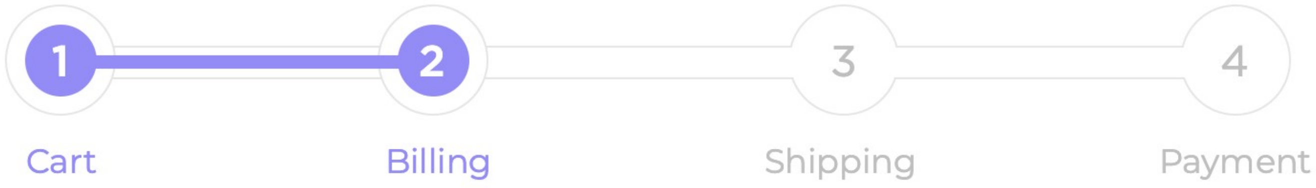
Each step is a "breadcrumb" and the full path is their clickstream session.


ELECTRONICS
MEN
WOMEN
BABY & KIDS
HOME & FURNITURE
BOOKS & MEDIA
AUTO & SPORTS
Home > Computers > Audio Players > Home Audio > Speakers
Location Based Breadcrumb
SPEAKERS
Attribute Breadcrumb
SHOWING 35 IN: JBL × BLUETOOTH ×
Search within Speakers
Popular
High Price
Low Price
New
Shortlist

Cart > Billing > Shipping > Payment

Cart
Billing
Shipping
Payment

Cart / Billing / Shipping / Payment


1
2
3
4
Cart
Billing
Shipping
Payment

Sample Clickstream Dataset

A sample clickstream dataset for a single user session on an e-commerce website

Timestamp	User ID	Page Visited	Action	Referrer	Product ID
2025-05-03 10:00:00	U123	Homepage	Page View	-	—
2025-05-03 10:00:05	U123	Search Page	Search	Homepage	—
2025-05-03 10:00:10	U123	Product Detail Page	Click	Search Page	P456
2025-05-03 10:00:30	U123	Cart Page	Add to Cart	Product Page	P456
2025-05-03 10:01:00	U123	Checkout Page	Checkout	Cart Page	P456

Data Includes:

- Timestamps of each action
- User identifiers (cookies, session IDs)
- Page URLs and referrers
- Action types (view, click, scroll, add-to-cart, etc.)
- Device/browser info
- Session IDs

Importance

Clickstream data is widely used in data science, analytics, and marketing to:

- Understand user behavior and session flow
- Identify where users drop off (conversion funnel analysis)
- Personalize content and product recommendations
- Optimize website layout and navigation
- A/B test different designs or promotions

Problem

Apply full Hadoop ecosystem (MapReduce, HDFS, YARN, Sqoop, and Flume, etc.) for log analysis and customer behavior analytics in an e-commerce company using clickstream logs and transactional databases to improve product recommendations, marketing campaigns, and website performance optimizations under following data volume, velocity, and consistency constraints

OR

You are working as a Data Engineer at a large e-commerce company. Your team has been tasked with building a big data pipeline to analyze customer behavior from two main data sources:

1. Clickstream Logs: Generated in real-time (~500GB/day) from 1000+ web servers.
2. Transactional Data: Stored in a MySQL database, containing user profiles, orders, and product details.

The company wants to generate insights such as:

- Session paths and customer journeys.
- Product recommendation inputs.
- Conversion funnel analysis.

However, the project must satisfy the following operational and technical constraints:

- Logs must be ingested in near real-time.
- MySQL database has a daily export limit of 10GB.
- Personal Identifiable Information (PII) like email and phone must be masked.
- The Hadoop cluster has only 20 nodes.
- MapReduce jobs must complete in under 2 hours.
- Log files may arrive with up to 4 hours delay.
- Data must be securely stored and efficiently queried later.

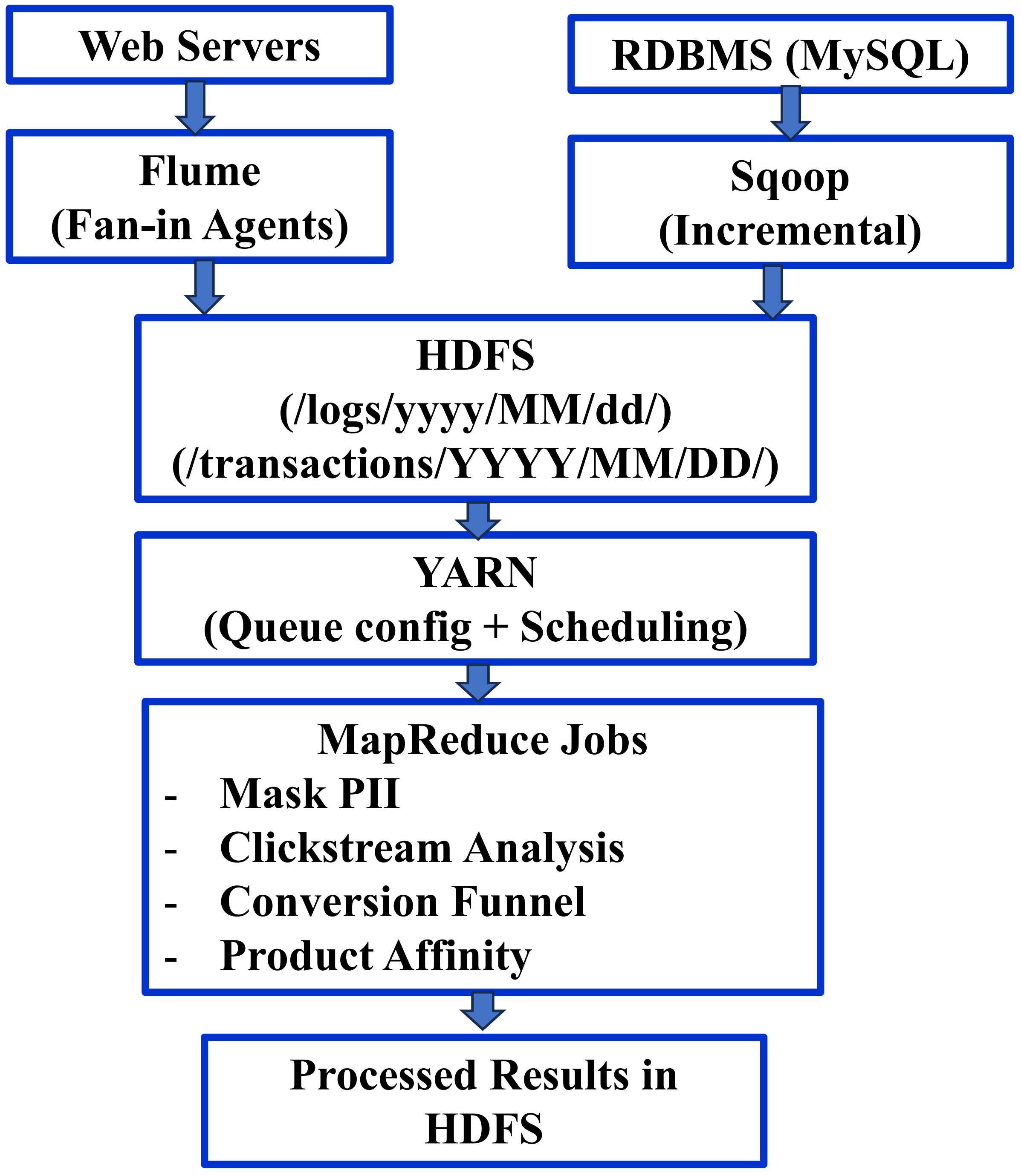
Develop a realistic end-to-end architecture and data flow using the following Hadoop ecosystem tools:

Flume, Sqoop, HDFS, YARN, MapReduce. Optionally, you may also consider Hive, Pig, or Spark if needed.

Explain

1. How will you design the data ingestion strategy under these constraints?
2. How will you ensure PII compliance during ingestion or processing?
3. How would you organize data in HDFS for efficient processing and late-arriving logs?
4. How will you structure and tune your MapReduce jobs to meet performance and resource constraints?
5. What additional techniques would you apply to ensure the architecture is scalable and fault-tolerant?

Architecture Diagram



1. Ingestion Strategy

Clickstream Logs:

- Use Apache Flume with a multi-agent, fan-in topology.
- Configure Memory Channel with File Channel failover for reliability.
- Sink logs to HDFS in compressed format (e.g., Avro, Parquet).
- Partition HDFS directory by event_date instead of ingestion time.

Transactional Data:

- Use Sqoop with incremental imports (based on last_updated column or auto-incrementing ID).
- Schedule Sqoop jobs during off-peak hours to avoid overloading the RDBMS

2. PII Masking / Privacy

- Use a Flume interceptor to mask or hash PII fields like email and phone.
- Alternatively, run a Map-only MapReduce job immediately after ingestion to clean the data before main processing.

3. HDFS Storage Layout

- Use time-based partitioning: /clickstream_logs/event_date=YYYY-MM-DD/
- Store structured data in columnar format (Parquet) to improve query efficiency.
- Maintain separate folders for raw, processed, and cleaned data.

4. MapReduce Job Optimization

- Use combiners and secondary sorting to reduce shuffle overhead.
- Limit number of reducers; optimize by analyzing intermediate key cardinality.
- Tune YARN container memory (mapreduce.map.memory.mb, mapreduce.reduce.memory.mb).
- Use counters to monitor job progress and delays.

5. Scalability & Fault Tolerance

- Implement retry logic and checkpointing in ingestion jobs.
- Handle late-arriving data with backfill jobs or event-time partitioned storage.
- Optionally use Hive for ad hoc querying and dashboards.
- Use workflow orchestrators like Oozie or Apache Airflow for job scheduling.