BUSINESS INTELLIGENCE AND DATA ANALYTICS
– WE PREDICT THE FUTURE –

University of Petra

Institute of Analytics

University of Petra

307401
Big Data and Data Warehouses

Introduction to Hadoop

Apache Hive

What is Hive?

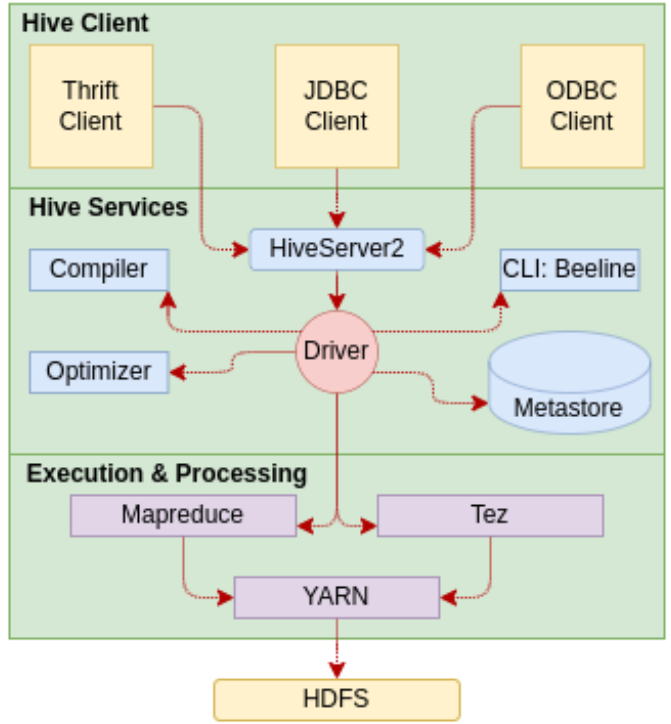
- Hive is a data warehouse system built on top of Hadoop.
- It allows querying and managing large datasets using a SQL-like language called HiveQL.
- Created by Facebook to simplify complex MapReduce programming for analysts.

Why Hive?

- Writing raw MapReduce code in Java is complex and time-consuming.
- Hive allows users to write declarative queries in HiveQL.
- Hive compiles these queries into efficient MapReduce tasks.

Hive Architecture

- Metastore: Stores metadata about Hive tables (schemas, locations, etc.).
- Driver: Manages the lifecycle of a HiveQL query and its execution plan.
- Compiler: Translates HiveQL into MapReduce jobs to run on Hadoop.



Real-World Use Case: Marketing Campaign Analysis

Company Example: NETFLIX

Objective: **Analyze customer behavior** to evaluate marketing campaign effectiveness.

Data Source: User activity logs stored in HDFS, including views, clicks, watch duration, and subscription status.

Data Structure (Hive Table Schema):

```
CREATE TABLE user_activity (
  user_id STRING,
  timestamp TIMESTAMP,
  action STRING,        -- e.g., "play", "pause", "subscribe",
"click_ad"
  content_id STRING,
  device_type STRING,
  campaign_id STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Processing with Hive (HiveQL):
Perform aggregations:

- Total views per campaign
- Conversion rates from ad click to subscription
- Device-specific campaign performance

Sample Query:

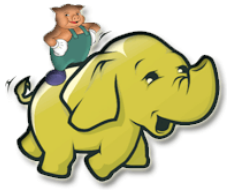
SELECT campaign_id, COUNT(*) AS views, SUM(CASE WHEN action = 'subscribe' THEN 1 ELSE 0 END) AS subscriptions FROM user_activity WHERE action IN ('click_ad', 'subscribe') GROUP BY campaign_id;

Outcome:

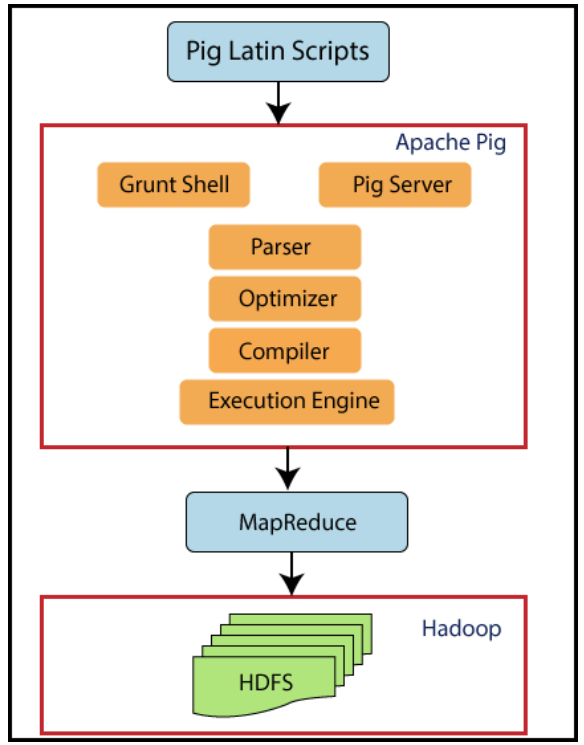
- Identifies high-performing campaigns and channels.
- Supports data-driven decisions for targeting and ad spend allocation.
- Let me know if you'd like this turned into a presentation slide or want a Hive use case from retail, telecom, or IoT.

Apache PIG – High Level SQL Like Language on top of Hadoop

- Run on Hadoop and translates Pig Latin scripts into MapReduce jobs.
- Created by YAHOO! to solve MapReduce challenges
- Pig Latin is a high-level data processing language that simplifies writing complex MapReduce code directly, this allows users to work with big data using simpler syntax compared to Map Reduce.
- Used for data transformations and analysis tasks over large datasets in Hadoop.
- **Architecture**:
  - **Pig Latin Scripts**: Define data transformation tasks.
  - **Pig Engine**: Executes these scripts on Hadoop.



APACHE PIG



https://www.analyticsvidhya.com/blog/2022/06/getting-started-with-apache-pig/

Hive vs Pig

Real-World Use Case: Web Log Processing for User Behavior Analysis

Company Example: *LinkedIn (used Pig before transitioning to Spark)*

Objective: Analyze web server logs to understand user navigation paths and session behavior.

Data Source:

- Daily web server log files stored in HDFS.
- Data Structure (Sample Log Fields):
- timestamp, user_id, session_id, page_url, referrer_url, device_type

Example Data:

2025-03-22T12:01:45Z, u12345, s7890, /jobs, /home, mobile

Processing with Apache Pig:

Load and Transform Logs:

```
logs = LOAD '/logs/2025/03/22'
    USING PigStorage(',')
    AS (timestamp:chararray, user_id:chararray, session_id:chararray,
      page_url:chararray, referrer_url:chararray,
device_type:chararray);
```

Group and Analyze Sessions:

```
grouped = GROUP logs BY session_id;
session_counts = FOREACH grouped GENERATE group AS session_id,
        COUNT(logs) AS page_views;
```

Filter High-Engagement Sessions:

```
engaged_sessions = FILTER session_counts BY page_views > 5;
```

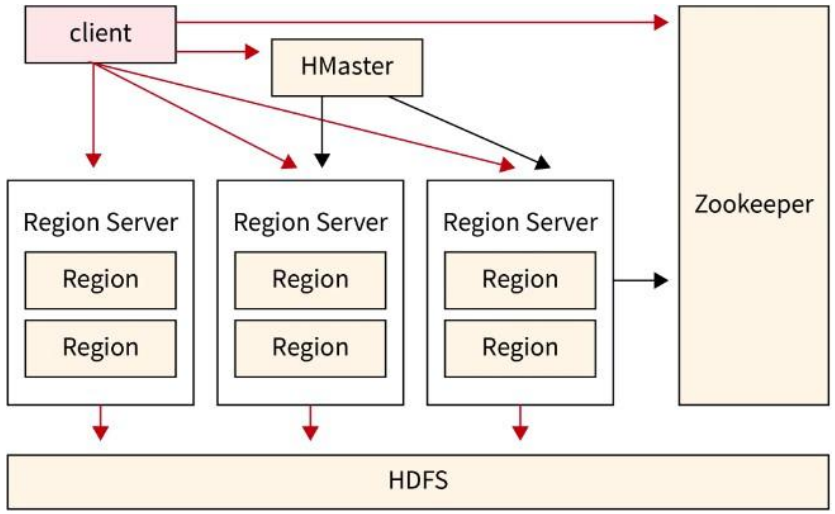
Outcome:

- Identifies highly engaged users and frequent navigation paths.
- Insights used for UX improvements and targeted content placement.

Apache HBase - Distributed Database on top of Hadoop

What is HBase?

- HBase is a distributed, column-oriented NoSQL database built on top of Hadoop HDFS.
- Designed for sparse, semi-structured, and unstructured data.
- Provides real-time read/write access to large datasets.
- Inspired by Google's Bigtable.



Apache HBase Architecture

Apache HBase Architecture

- **HMaster**: Manages the cluster.
- **RegionServer**: Handles read/write requests for regions of the database.
- **ZooKeeper**: Manages distributed coordination.

https://www.scaler.com/topics/hadoop/hbase-architecture/

Insert Data in HBase

- In HBase, we create tables and define column families.

create 'employees', 'personal', 'job'	**Create a table named employees with two families (personal and job)**

- Then we insert values

put 'employees', 'emp1', 'personal:name', 'Alice'	emp1 has only name in personal
put 'employees', 'emp2', **'personal:phone'**, '123456789'	emp2 has only phone in personal
put 'employees', 'emp3', **'job:salary'**, '60000'	emp3 has only salary in job, no personal column family

- Notice that must comply with column families when we insert data into the table.
- But we can use different column names within the column family e.g.

✅ **Schema-less?** Yes, **for columns** (qualifiers) – flexible and dynamic.

🧱 **Schema-fixed?** Yes, **for column families** – must be defined when the table is created.

Key Concepts

- Row Key: Unique identifier for each row (like a primary key).
- Column Family: Logical group of columns stored together on disk.
- Column Qualifier: Actual field name inside a column family.
- Timestamp: Each cell value is versioned by time.
- Cell: Combination of Row Key + Column Family + Qualifier + Timestamp → Value.

Each entry in HBase is stored as a **triplet (technically a quintuple):**

pgsql Copy Edit

```
(Row Key, Column Family, Column Qualifier, Timestamp) => Value
```

This means **each field you insert becomes a cell**, and there's no fixed schema for columns like in a relational table.

Example: Employee Data Internally in HBase

Let's say you added this:

bash Copy Edit

```
put 'employees', 'emp1', 'info:name', 'Alice Smith'
put 'employees', 'emp1', 'info:salary', '72000'
```

Internally, it looks more like:

Row Key	Column Family	Column	Timestamp	Value
emp1	info	name	1679553600000	Alice Smith
emp1	info	salary	1679553600000	72000

> HBase auto-generates timestamps unless you set them manually.

Real-World Use Case: Messaging Platform – Real-Time Message Storage & Retrieval

Company Example: *WhatsApp*

Objective: Store and retrieve billions of real-time chat messages with low latency and high throughput.

Data Source: User chat messages from mobile devices, synced to a backend service.

Data Structure (HBase Table Design):

Table: messages
Row Key: user_id + timestamp
Column Families:

- meta: sender_id, receiver_id, message_type
- content: message_type, attachments
- status: delivered, read, timestamp

Sample Record (in HBase):
Row Key: user123|2025-03-22T14:33:21Z
Column Family: meta
- sender_id: user123
- receiver_id: user456

Column Family: content
- message_text: individual

Column Family: status
- delivered: true
- read: false

Processing with HBase:

- Messages are written and read with low latency.
- Efficient for real-time features: chat history retrieval, unread message counts.
- Scales horizontally to billions of rows (messages) per day.

Outcome:

- Ensures seamless chat experience across devices.
- Supports real-time sync, delivery status tracking, and storage durability.

HBase vs Relational DB

Feature	HBase	Relational DB
Schema	Flexible (schema-less columns)	Fixed schema
Row Structure	Rows can have different columns	Rows have same set of columns
Access Pattern	Real-time random read/write	Typically, transactional
Query Language	API (Java, REST, Thrift)	SQL
Storage Model	Column-oriented, sparse	Row-oriented
Best Use Case	Time-series, logs, IoT, real-time	Structured, normalized data

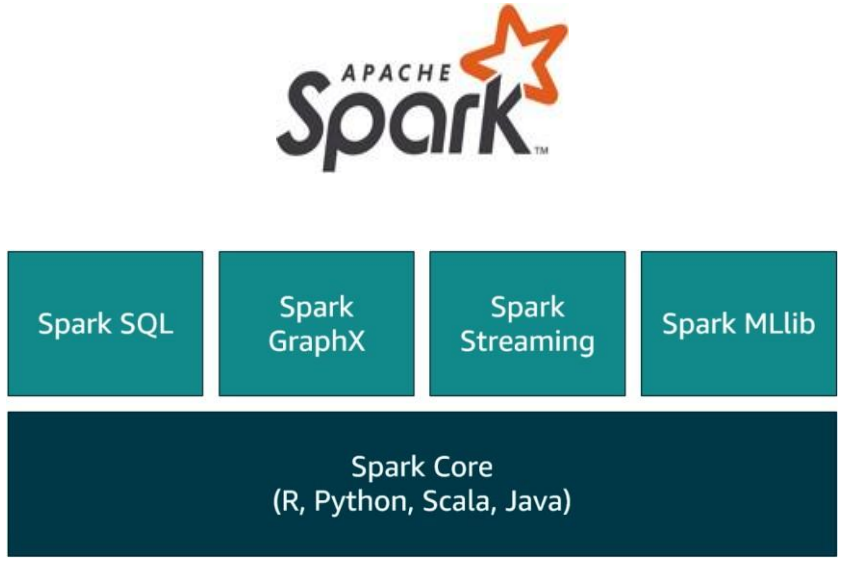
Apache Spark

Definition:

- Apache Spark is an open-source distributed data processing engine optimized for large-scale data processing.
- In-Memory Processing: Spark keeps data in memory between transformations, speeding up processing times significantly compared to disk-based MapReduce.
- Spark can handle both batch and real-time processing.
- Spark offers APIs for various languages (Java, Scala, Python, R).

Core Components:

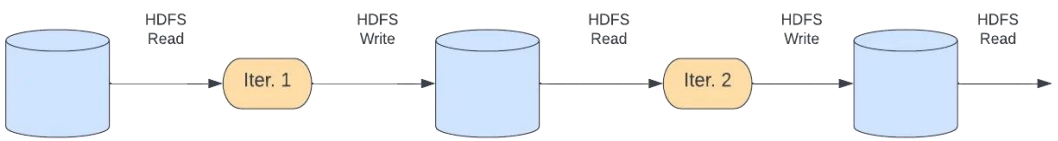
1. **Spark Core:** The foundation for Spark's API, handling basic data processing and distributing tasks across the cluster.
2. **Spark SQL:** Enables SQL-like queries on data stored in distributed datasets (DataFrames and Datasets).
3. **Spark Streaming:** Allows processing of real-time data streams, enabling near real-time analytics.
4. **MLlib:** Spark's machine learning library, offering tools for scalable machine learning.
5. **GraphX:** A graph processing library for large-scale graph computations.



MapReduce vs. Spark

MapReduce – First-Generation Framework

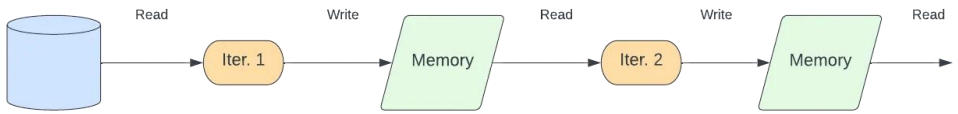
- A disk-based distributed processing system designed for large-scale, fault-tolerant batch computation.
- Follows a linear execution model:
  - Read data from HDFS
  - Apply map() function
  - Write intermediate data to disk
  - Shuffle and sort
  - Apply reduce() function
  - Write final output to HDFS
- Every phase involves reading from and writing to disk, which increases latency.
- Suitable for simple, one-pass data processing but inefficient for iterative tasks like machine learning or graph algorithms.
- Key Limitation: Disk I/O at every stage leads to poor performance for multi-stage workflows.



MapReduce

Apache Spark – Third-Generation Framework

- An in-memory distributed processing engine that improves on MapReduce's model.
- Uses Resilient Distributed Datasets (RDDs) to manage distributed data in memory.
- Supports chaining multiple transformations without writing intermediate results to disk.
- Writes to disk only when necessary (e.g., final output or fault recovery).
- Key Advantage: Significantly faster than MapReduce for iterative and interactive workloads.

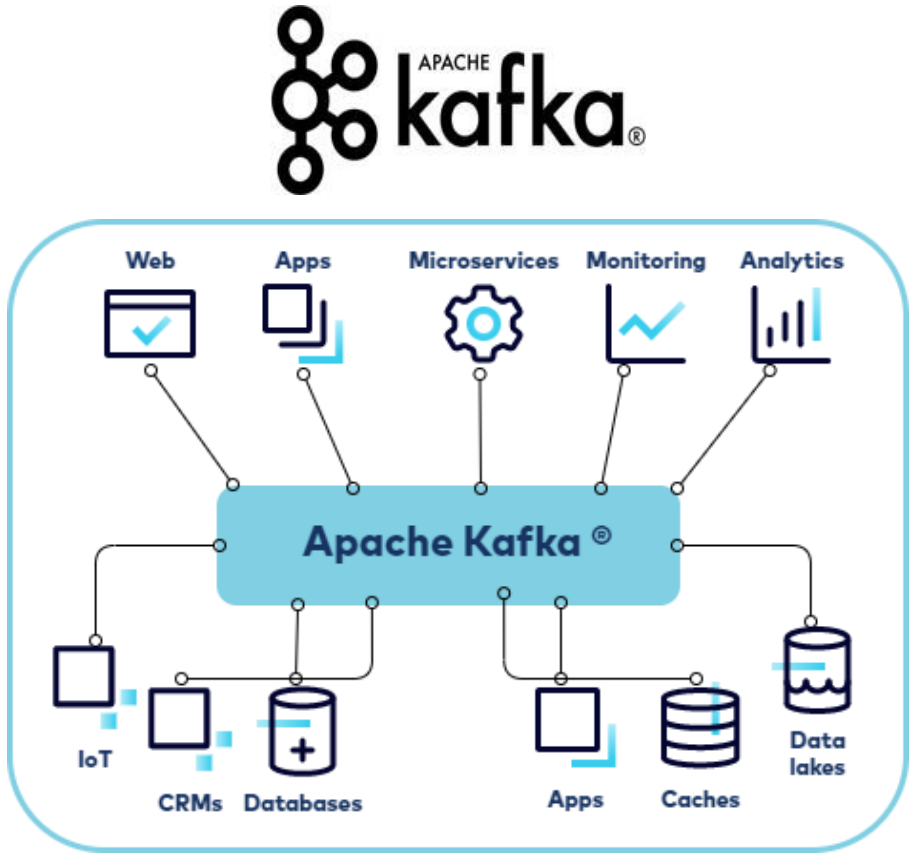


Apache Spark

https://www.redpanda.com/blog/data-stream-processing-spark-flink-ksqldb

Apache Kafka – Messaging System

- Apache Kafka is a distributed event streaming platform designed to handle high-throughput, real-time data feeds.
- Originally developed by LinkedIn, Kafka has become the backbone for data pipelines and real-time analytics in many large-scale systems.
- It serves as a message broker that allows various applications to produce and consume streams of data reliably, efficiently, with low latency and at scale.
- **Architecture**:
  - **Producer:** An application that sends data to Kafka topics.
  - **Consumer:** An application that reads data from Kafka topics.
  - **Broker:** A Kafka server that stores data and serves it to consumers.
  - Kafka clusters consist of multiple brokers.
  - **Topic:** A category or feed name to which records are published. Each topic can have multiple partitions, helping Kafka to parallelize data across the cluster.



Kafka in E-Commerce Order Processing

Scenario:

- An online store uses **Kafka** to manage orders, inventory, payments, and shipping in real time.

Process Flow:

1. **A customer places an order** on the storefront (website/app).
2. The order details are sent to the **Kafka orders topic**.
3. The **Order Processing Service** consumes the order, checks inventory, and verifies payment.
4. If items are in stock, the order is confirmed, and a message is sent to the **Inventory & Payment topics**.
5. Once the payment is confirmed, the **Shipping Service** picks up the order details from Kafka and generates a shipment.
6. The **Customer Notification Service** listens for updates and sends notifications.

Kafka in E-Commerce Order Processing – Extended Example

1. Order Initiation - The Story Begins

- **What's happening:** Sarah is shopping on your e-commerce website. She's added a few items to her cart - a blue sweater, some headphones, and a coffee mug. After reviewing everything, she clicks the "Place Order" button.
- **Behind the scenes:** The moment Sarah hits that button, your website (the Online Storefront) springs into action. It's like a cashier at a store who just received Sarah's order and now needs to tell everyone else in the company about it.
- **The Handoff:** The website doesn't try to handle everything itself. Instead, it creates a detailed "order slip" with all of Sarah's information:
  - A unique order number (like #ORD-12345)
  - Sarah's details (name, email, shipping address)
  - What she ordered (blue sweater size M, wireless headphones, ceramic mug)
  - How many of each item
  - The total amount she needs to pay ($89.97)
  - This is like posting a message on a company bulletin board that says "New order just came in!" - except it's instant and digital.
- **The Announcement:** The website then puts this order slip into a digital "mailbox" called the orders topic in Kafka.
- **Why this matters:** By putting the order into this shared mailbox, the website is essentially saying "Hey, Order Processing team, we've got a new order for you to handle!" The website's job is done - it just needs to pass the baton to the next person in line.

Kafka in E-Commerce Order Processing – Extended Example

2. Order Validation & Processing
Trigger: New order event received
Actor: Order Processing Service
Actions:
Validates order details
Checks customer information
Requests inventory check
Decision Point: Is order valid?
Valid: Continue to inventory check
Invalid: Send failure notification

3. Inventory Check & Reserve
Trigger: Order validation complete
Actor: Order Processing Service → Inventory Management Service
Action: Publishes inventory request to inventory topic
Response: Inventory Management Service confirms/denies availability
Decision Point: Items available?
Available: Reserve items, continue to payment
Unavailable: Cancel order, notify customer

4. Payment Processing
Trigger: Inventory reserved
Actor: Order Processing Service → Payment Gateway
Action: Initiates payment processing
Response: Payment Gateway publishes result to payments topic
Decision Point: Payment successful?
- **Success:** Mark order as paid, proceed to fulfillment
- **Failure:** Release inventory, cancel order, notify customer

5. Shipping Preparation
Trigger: Payment confirmed
Actor: Order Processing Service → Shipping Service
Action: Publishes shipping request to shipping topic
Data: Order details, shipping address, delivery preferences
Result: Shipping Service creates shipment and assigns tracking number

6. Warehouse Fulfillment
Trigger: Shipping request received
Actor: Warehouse System
Actions:
Picks items from inventory
Packages order
Updates inventory levels (publishes to inventory topic)Hands off to shipping provider

E-commerce Order Processing Flow

7. Shipping & Delivery Tracking

Trigger: Package dispatched

Actor: Shipping Provider

Actions:

- Updates delivery status
- Publishes tracking updates to shipping topic
- Provides delivery confirmation

8. Customer Notifications (Parallel Throughout)

Trigger: Various status changes

Actor: Customer Notification Service

Listens to: notifications topic

Actions:

- Order Confirmed
- Payment processed
- Order shipped
- Out for delivery
- Delivered
- **Channels:** Email, SMS, push notifications

Apache Flink

What is Apache Flink?

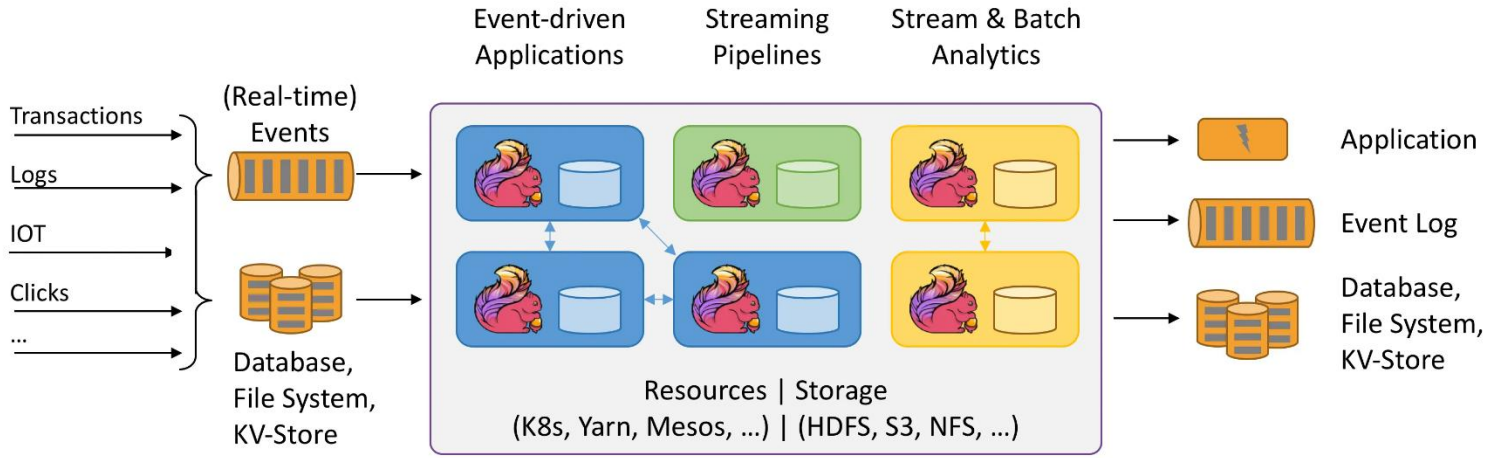
- Open-source, distributed stream & batch processing system
- Originally developed by the Stratosphere project, now under Apache Software Foundation
- Processes high-velocity data with low latency, exactly-once semantics, and strong fault tolerance

Architecture

- **Sources** – Data ingestion from Kafka, databases, files
- **Transformations** – Stateful operations like filtering, aggregations, and joins
- **Sinks** – Writing processed data to Kafka, databases, or data lakes
- **Stateful Processing** – Uses checkpoints and savepoints for fault tolerance
- **Event Time Processing** – Handles late-arriving data with watermarks

Use Cases

- Real-time analytics – Log analysis, anomaly detection, monitoring
- Event-driven applications – Dynamic data enrichment, alerting
- ETL pipelines – Continuous data processing and transformation
- Machine learning – Real-time model inference and training



Apache Oozie

Apache Oozie is a **workflow orchestration system** designed to manage and coordinate Hadoop jobs in a **sequence** or **directed acyclic graph (DAG)**. It provides **end-to-end workflow management** for big data pipelines, making it easier to automate complex multi-step data processing tasks.

Key Features

- **Workflow Scheduling:** Define and run workflows consisting of multiple Hadoop jobs (e.g., MapReduce, Hive, Pig, Sqoop).
- **Dependency Management:** Manage job dependencies and ensure tasks run in the correct order.
- **Trigger Support:** Supports both **time-based triggers** (e.g., hourly, daily) and **data availability-based triggers**.
- **Error Handling & Retry:** Automatically retries failed jobs and handles exceptions through recovery mechanisms.
- **Flexible Integration:** Works with various Hadoop components and supports shell scripts, Java programs, and custom actions.

Common Use Cases

- Automating **ETL pipelines** (Extract, Transform, Load)
- Managing **daily batch processing** jobs
- Scheduling **data ingestion workflows** using Sqoop or Flume
- Orchestrating **multi-tool data flows** (e.g., Hive → Pig → MapReduce)

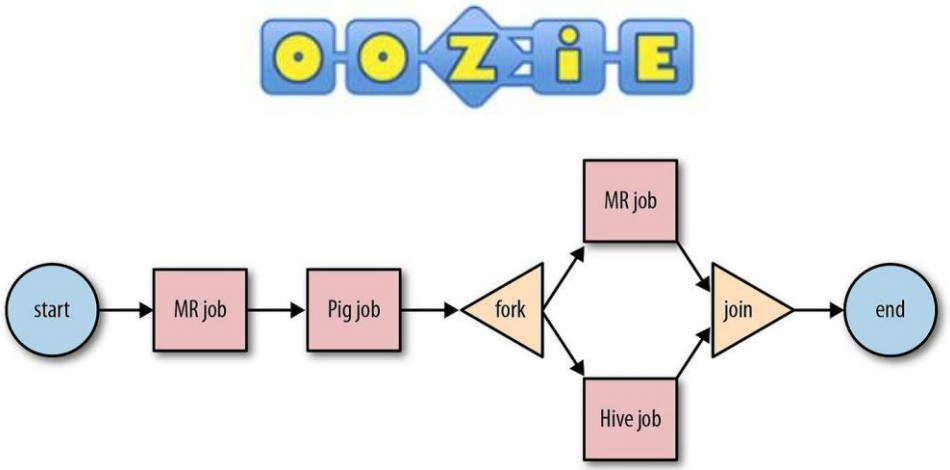
Real-Life Use Case

Enterprise Data Warehouse Refresh:

A retail company uses Apache Oozie to automate the **nightly refresh of their enterprise data warehouse**. The workflow includes:

1. **Importing sales data** from relational databases into HDFS using Sqoop.
2. **Cleansing and transforming data** using Hive queries.
3. **Aggregating key metrics** with MapReduce.
4. **Loading results** into a Hive-based reporting table.
5. **Triggering email alerts** if a job fails or data thresholds are not met.

This allows the analytics team to have **up-to-date insights each morning**, without manual intervention.



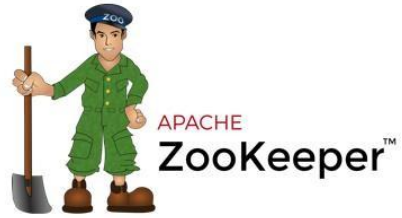
Apache Zookeeper

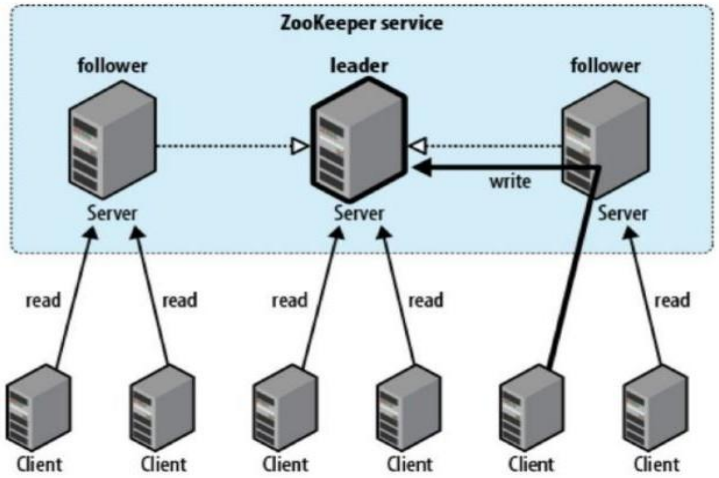
Apache ZooKeeper is a centralized service designed for coordinating distributed applications.

ZooKeeper plays a critical role in the Hadoop ecosystem and other distributed systems by helping to maintain the distributed state and providing a reliable way for services to work together.

Key Features of Apache ZooKeeper

1. **Centralized Configuration Management**: ZooKeeper stores configuration data in a centralized manner, allowing distributed applications to access and modify configuration settings easily.
2. **Naming Service**: It provides a naming registry for distributed services, allowing them to find each other and manage resources dynamically.
3. **Synchronization**: ZooKeeper helps synchronize distributed processes, ensuring they work in coordination and without conflicts.
4. **Group Management**: Supports the creation and management of groups, enabling services to join and leave groups dynamically.
5. **Watchers**: ZooKeeper can set watchers on nodes to get notified of changes, allowing applications to react to changes in state.





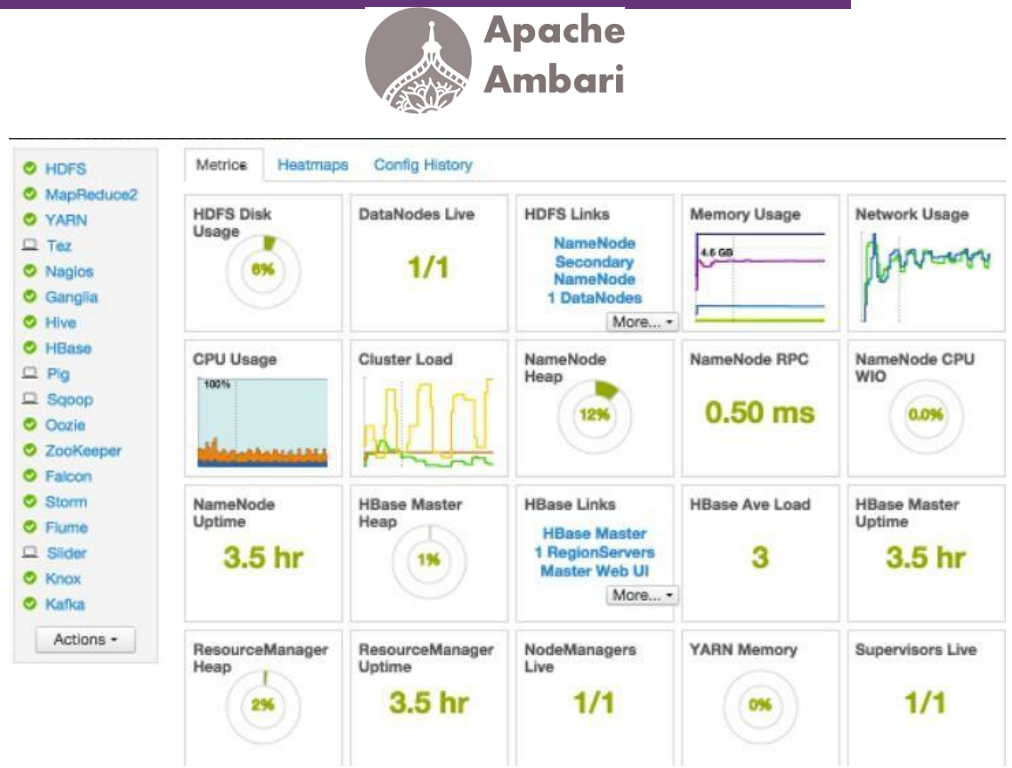
Apache Ambari

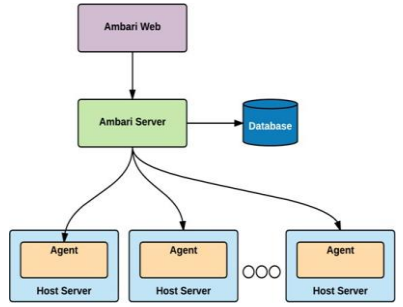
Apache Ambari is a web-based tool designed for provisioning, managing, and monitoring Apache Hadoop clusters.

It provides a user-friendly interface that simplifies the management of complex Hadoop environments, making it easier for administrators and operators to deploy and manage various components of the Hadoop ecosystem.

Key Features of Apache Ambari

- **Cluster Provisioning**: Ambari simplifies the installation and configuration of Hadoop clusters, allowing users to easily add or remove nodes and services through a guided installation wizard.
- **Service Management**: Users can start, stop, and restart various Hadoop services (such as HDFS, YARN, Hive, HBase, etc.) through the Ambari interface.
- **Monitoring and Alerts**: Ambari provides dashboards for real-time monitoring of cluster health, service metrics, and resource usage. It also supports setting up alerts for various cluster events or metrics.
- **Configuration Management**: It allows for easy management of configuration settings for Hadoop services, including the ability to apply configurations in a centralized manner.
- **User Management**: Administrators can manage users and their permissions, ensuring secure access to cluster resources and configurations.
- **Integration with Ambari Metrics and Alerts**: Supports the collection and visualization of metrics from different Hadoop components, allowing for comprehensive monitoring and alerting.





Main Hadoop Components

Category / Component	Description
Storage Components	
HDFS	Distributed file system for scalable and fault-tolerant data storage.
HBase	NoSQL database for real-time read/write access to large datasets.
Phoenix	SQL layer on top of HBase for low-latency OLTP and analytics.
Batch Processing	
MapReduce	Traditional batch processing model for parallel computation on large data.
Tez	DAG-based execution engine; faster alternative to MapReduce.
Hive	SQL-like interface for querying large datasets, often runs on Tez or Spark.
Pig	Dataflow scripting platform using Pig Latin for batch data processing.
Impala	MPP (Massively Parallel Processing) SQL query engine for real-time, interactive querying on Hadoop data.
Real-Time / Stream Processing	
Spark Streaming	Micro-batch stream processing engine within Apache Spark.
Flink	True stream processing engine with high throughput and low latency.
Storm	Distributed real-time computation system for event-driven processing.
Data Ingestion & Integration	
Sqoop	Imports/exports structured data between Hadoop and relational databases.
Flume	Collects and transports large volumes of log or event data.

Main Hadoop Components

Category / Component	Description
Security & Governance	
Ranger	Centralized security for authorization, policies, and auditing.
Knox	Gateway that provides perimeter security and REST API access to Hadoop.
Atlas	Metadata management and data governance framework.
Machine Learning & Search	
Mahout	Scalable machine learning library for classification, clustering, etc.
Solr	Enterprise search platform; supports full-text search on Hadoop data.
Cluster Coordination	
YARN	Resource management layer for job scheduling and cluster resource utilization.
Zookeeper	Coordination service for configuration, synchronization, and leader election.
Serialization & File Formats	
Avro	Row-based data serialization framework with schema support.
Parquet	Columnar storage format optimized for performance and compression.
ORC	Columnar storage format optimized for Hive; enables efficient querying.
UI, Workflow & Dev Tools	
Oozie	Workflow scheduler to orchestrate complex Hadoop jobs.
Ambari	Web UI for provisioning, monitoring, and managing Hadoop clusters.
Hue	Web-based user interface for querying and interacting with Hadoop tools.
Zeppelin	Interactive data analytics notebook that supports Spark, SQL, and more.