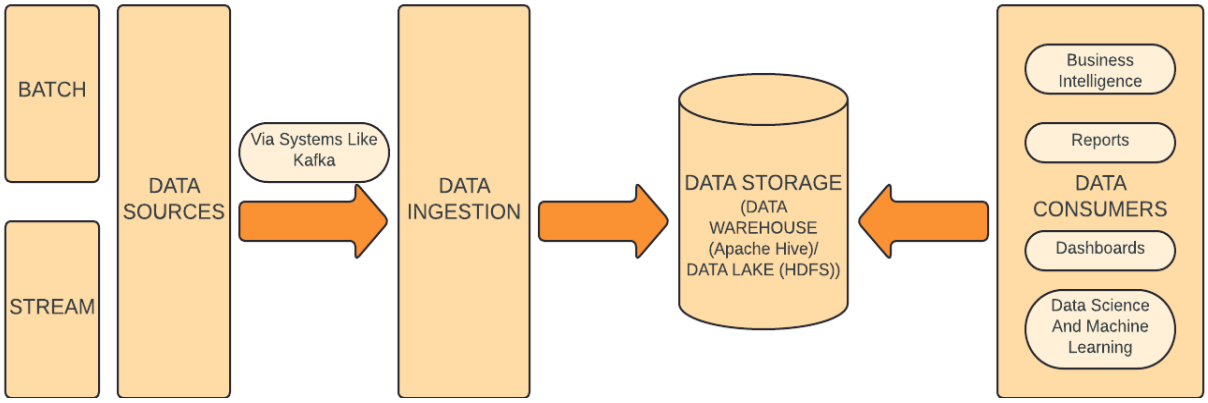
In the last lecture, we learned about what defines big data. In this lecture, we will understand the high-level architecture of big data technologies and how they fit into the ecosystem.



Data Platform Architecture

1. Data Sources

If we are going to learn about data engineering, the first thing that we should discuss is Data itself. Data Sources are categorized either by type of data or delivery of data.

Data categorization based on the type of data is the same as we have already discussed in a variety of big data. Just to summarize that again.

- **Structured Data** includes data that is highly organized and easily searchable, like data stored in relational databases such as MySQL or PostgreSQL.
- **Semi-Structured Data** doesn't fit neatly into the structure of a relational database but still contains some organizational properties, like JSON or XML files.
- **Unstructured Data** refers to data that doesn't have a predefined data model or is not organized in a predefined manner. Examples include log files, social media feeds, videos, and images.

Data sources can also be categorized by the way they are delivered i.e RealTime or Batch

- **Real-Time Streaming Data**: Real Time Streaming data as the name suggests is generated continuously from various sources like sensors, logs, user

interactions, financial transactions, and social media activities. This data arrives in small, often irregular, chunks that must be processed immediately.

One of the defining features of real-time streaming data is the need for low-latency processing. Data is processed and analyzed within milliseconds or seconds of its arrival, enabling immediate reactions and decisions.

- **Batch Data**: Batch data is not a different data. But it's the data that is not required immediately and is mainly used for analytical purposes sometime later. The data for batch processing is typically gathered from various sources over some time. These sources can include databases, file systems, logs, or any other form of data storage. It might be also possible that real-time data is later treated as batch data to process the data over a given time range.

Let's consider a customer of Flipkart orders something on the website during their BBD Sale, now the leadership of Flipkart might need to look at this data in real-time to see how the sale is going, now the same data is required even when the sale is over to maybe analyzed complete sales figure, auditing, monthly sales, etc... So, in this case, same data is being treated as realtime and batch at different time.

2. Data Ingestion

Once the data is generated, the next critical step is ensuring it reaches the big data system where it can be analyzed and used. This process is known as **data ingestion**. As we have seen there are 2 types of data - realtime and batch, so they have respective ingestion processes that are **real-time processing (streaming)** or **batch processing**.

Batch Processing

Batch Processing refers to the processing of large volumes of data that have been collected over a period of time. In this approach, data is collected, stored, and then processed in batches, typically at scheduled intervals.

Key Characteristics:

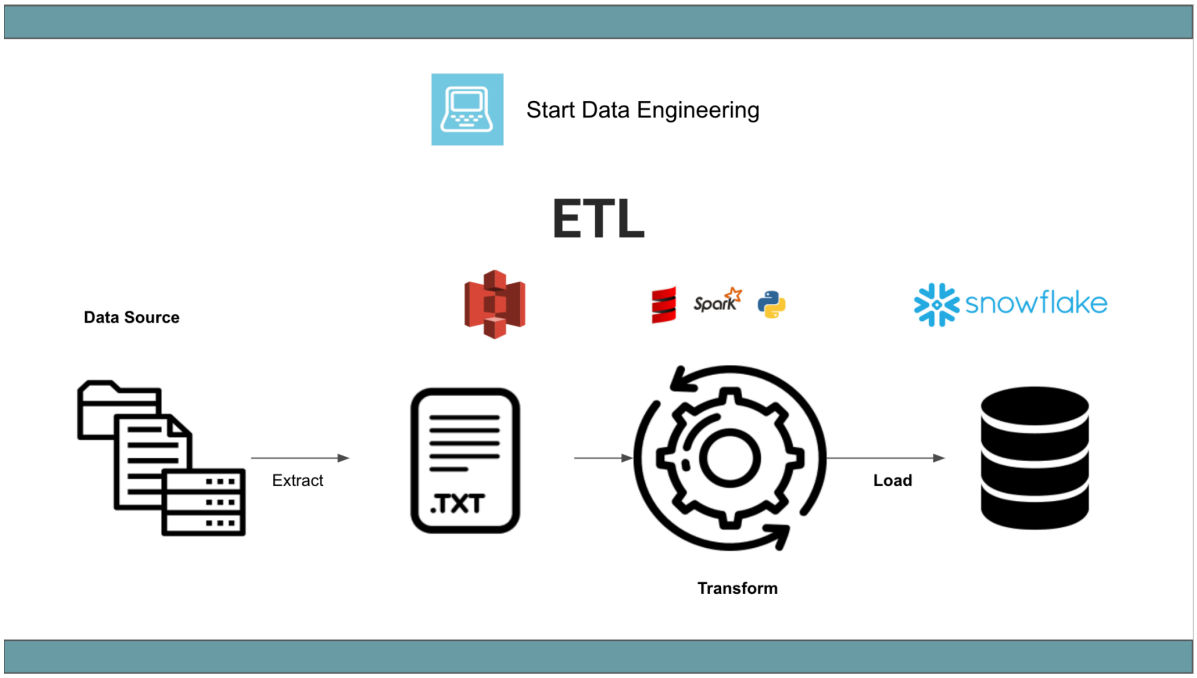
Lets forget data for a minute and try to understand the characteristics of batch analytics using a simple scenario in which a large company with thousands of employees processes payroll at the end of every month. Employee work hours, bonuses, and deductions are tracked throughout the month, and at the end of the month, the payroll system processes all of this data in one go to generate paychecks for every employee.

- **Volume:** Batch processing can handle **large volumes of data at once**. In this case, the payroll system must process salary calculations for thousands of employees, factoring in work hours, overtime, bonuses, and deductions, all at the same time. The company waits until the end of the month to process all employee payroll data in one large batch. This allows the system to handle a large volume of data efficiently, ensuring paychecks are calculated and distributed for every employee in a single run.
- **Latency:** Batch processing involves **a delay between data collection and processing**. The system collects data throughout the month (work hours, bonuses, etc.), but processing happens only at the end of the month. This delay (latency) is inherent to batch processing. Although employee work hours and other payroll-related data are recorded daily, the payroll is processed only once at the end of the month. This means there's a **latency** of 30 days between data collection (daily logging of work hours) and processing (generating paychecks).
- **Throughput:** Batch processing is designed for **high throughput**, meaning it can process large amounts of data quickly because it processes everything in one go. Once the month ends, the payroll system runs a batch process to calculate salaries for all employees. Instead of calculating paychecks for each employee individually, the system processes thousands of paychecks at once, ensuring high throughput. The system completes the payroll process for the entire organization within a short period.
- **Complexity:** Batch processing can handle complex calculations and aggregations, as it has the entire dataset available at once. Payroll processing involves calculating work hours, overtime, bonuses, tax deductions, and other factors, which are complex to calculate for each employee. Since the system has access to all the necessary data at the end of the month, it can run these complex calculations in a single batch for each employee, applying tax rules, deductions, and bonuses accurately.

More Use Cases:

- **Data Warehousing:** Loading and processing large datasets for analytics and reporting.
- **Log Analysis:** Analyzing server logs to generate reports on system performance.

Technologies used:



- **Apache Hive on MR:** A classic example of a batch processing framework. It divides the data into chunks, processes them in parallel, and then aggregates the results.
- **Apache Spark:** While Spark is known for its speed, it also excels at batch processing with its ability to process data in-memory.

Example Scenario:

- Imagine a financial firm that needs to analyze daily transaction data to generate end-of-day reports. They collect all the data throughout the day and run a batch job overnight to process and analyze the transactions.

Stream Processing

Stream Processing deals with the continuous and real-time processing of data as it arrives. Instead of waiting for data to accumulate, stream processing systems process each data point or small batch of data immediately upon arrival.

Key Characteristics:

Similar to what we tried in batch processing, let's consider a scenario of the online banking system that processes thousands of transactions every second. The bank uses a stream processing system to detect potentially fraudulent activities (like unusual login attempts or suspicious transactions) in real time. The system continuously monitors transactions and reacts immediately to prevent fraud.

- **Real-Time Processing:** Stream processing allows for **near-instantaneous analysis** and response to data as it is received. Unlike batch processing, data doesn't accumulate over time, instead, it is processed as soon as it arrives. In the fraud detection system, every time a customer makes a transaction, the system instantly analyzes the transaction for signs of fraud. If a suspicious pattern is detected (e.g., an unusually large transaction or access from an unfamiliar location), the system can immediately trigger an alert or block the transaction in real time.
- **Low Latency:** Stream processing has **minimal delay (low latency)** between data generation and processing. This allows systems to react quickly to changes or events as they happen. When a customer initiates a transaction, the fraud detection system processes the data immediately, within milliseconds. Because of this low latency, the bank can take action (such as sending an SMS alert or blocking the transaction) almost instantly, reducing the likelihood of fraudulent activity going undetected.
- **Scalability:** Stream processing systems need to be **scalable** to handle varying volumes of data, especially when dealing with high-velocity data streams (such as millions of transactions per second). As the number of online banking transactions increases during peak hours (like payday), the stream processing system must scale to handle the surge in data flow. The bank's fraud detection system scales horizontally by adding more servers or resources to ensure it can process a higher volume of real-time data without slowing down or missing any transactions.
- **Event-Driven:** Stream processing is often **event-driven**, meaning that each data event (such as a login attempt or a transaction) triggers some specific action or processing logic in the system. In the fraud detection system, every transaction is treated as an event. When a transaction occurs, it triggers a series of checks to validate whether the transaction is legitimate or suspicious. For example, if the transaction exceeds a certain limit, it triggers an additional layer of verification, like sending a one-time password (OTP) to the customer.

More Use Cases:

- **Real-Time Analytics:** Monitoring system performance or user activity in real time to make instant decisions.
- **IoT Data:** Processing data from sensors and devices in real-time to monitor conditions and control actions.

Technologies:

- **Apache Kafka:** Often used as a messaging system to handle high-throughput real-time data streams.
- **Apache Flink:** A stream processing framework designed for stateful processing and event-driven applications.
- **Apache Spark Streaming:** An extension of Spark for processing live data streams with the same API used for batch processing.

Example Scenario:

- Consider a streaming service like Netflix, which processes user interactions in real time to update recommendations and optimize the viewing experience. As users watch shows, data is immediately processed to suggest new content.

3. Data Storage

Now let's understand data warehouse and data lake. Imagine you have a huge collection of books. A **data warehouse** is like a well-organized library where every book is carefully categorized and placed on the right shelf. You can quickly find what you need because everything is neatly arranged and labeled. It's perfect for when you want to look up something specific or analyze certain types of information.

On the other hand, a **data lake** is like a giant storage room where all kinds of books, documents, and papers are just thrown in. There's no specific order, but you can store anything, even if you're not sure how you'll use it later. It's great for keeping everything, even the messy or unstructured stuff, but it might take more effort to find exactly what you're looking for.

So, a data warehouse is structured and organized for quick access, while a data lake is more flexible, storing everything in its raw form, ready for future use.

- Data Lake: Storage for raw, unstructured, or semi-structured data (e.g., AWS S3, Azure Data Lake, HDFS).
- Data Warehouse: Central repository for structured data optimized for analytical queries (e.g., Amazon Redshift, Google BigQuery, Apache Hive).

Now lets take a loot at one of the Datalake and Data Warehouse technologies

HDFS (Data Lake)

HDFS is among the most common data lakes used. To understand how HDFS works, let's begin by considering a simple scenario where we have a single host—a computer or server—with 1 TB of hard disk space. This setup works fine when our

data needs are minimal. However, as our data requirements grow, we may find that 1 TB of storage is no longer sufficient.

Vertical Scaling

To address this, one option is to increase the storage capacity of our existing system by adding more hard disk space. This approach is known as **vertical scaling**. Vertical scaling involves upgrading the resources of a single machine—like adding more disk space, memory, or processing power.

However vertical scaling has its limits. Eventually, you'll reach a point where you can't add more resources, or it becomes too costly or inefficient. This is where **horizontal scaling** becomes essential.

Horizontal Scaling

In **horizontal scaling**, instead of adding more resources to a single machine, you add more machines, or **nodes**, to your system. Imagine you now have 100 nodes, each with 1 TB of hard disk space. Collectively, these 100 nodes form a **cluster** with a total of 100 TB of usable storage.

Now, whenever a data request comes into this cluster, the data can be stored on any of the 100 nodes. When you need to retrieve the data, you simply request it from the cluster.

This is the basis of how data is stored in HDFS. Now what the complete architecture looks like, we will discuss in subsequent lectures.

Apache Hive (Data Warehouse)

Apache Hive is one of the most common data warehouse infrastructure built on top of Hadoop's Distributed File System (HDFS) which we have just discussed. Hive is not just a ware house but also a processing engine which are going to see in next section. What makes Hive super useful is that It enables users to query and manage large datasets stored in HDFS using a SQL-like language called HiveQL. Hive organizes data into databases, tables making it easier to perform complex analytical queries. So once data is stored in Hive, even a person without much context of Big data systems can easily interact with Hive and get the required data for analysis. We are going to discuss hive's architecture in detail in upcoming lectures.

Data Processing

Now we have learnt about data, its source, its entry into big data world and where its stored, but now comes a very important question. How its stored and how its processed later for analysts and others to use.
Data in its raw state is not much of use as it might not hold value and might be in a state which doesn't make it fit to consume. Consider an example of Flipkart, where order details are stored in MySql db. Details of order like user name, contact, email , products ordered, price of product, pincode etc are stored. Now hen storing this data in big data systems we won't want data like phone number to be stored and when storing product details in big data systems , we would want to know the category of product, its other metadata so that it can be used for further analysis>

Now we need to transform our data during ingestion phase itself, to make sure that we have right data stored in our tables. Now as data is too large due to order volume as Flipkart that we would need specialized systems to get this information from MySQL like database, process the data and then store it properly, that's where data processing in Big data comes into the picture. The technique of getting this cleaned data that we just described here is ETL. So what is ETL ?

ETL Tools (Extract, Transform, Load)

ETL stands for **Extract, Transform, Load**, and it is a crucial process in data management, especially for e-commerce platforms. Let's break down each step of the ETL process using an example from an e-commerce platform like Flipkart.

Extract

The first step of this process is extracting data from the target sources that are usually heterogeneous such as an SQL or NoSQL database, a cloud platform, or an XML file.

It refers to the process of collecting data from various sources. For an e-commerce platform, data might come from:

- **Order databases**: Information about customer orders, including items purchased, prices, quantities, and shipping details.
- **Product catalogs**: Data about product details, stock levels, and pricing.
- **Customer databases**: Information about customer profiles, browsing history, and past purchases.
- **Web logs**: User interaction data, such as page views, clicks, and search queries.

Example: Suppose Flipkart wants to analyze the buying patterns during their annual Big Billion Days sale. The extraction phase would involve pulling data from the order database, product catalog, and web logs for the duration of the sale.

Transform

It involves cleaning, structuring, and enriching the extracted data to make it usable for analysis. This step may include:

- **Data cleaning**: Removing duplicates, handling missing values, and correcting errors.
- **Data normalization**: Ensuring data from different sources follow the same format and structure.
- **Aggregation**: Summarizing data, such as calculating total sales per product or per customer.
- **Enrichment**: Adding additional data, such as categorizing products or linking customer demographics.

Example: After extracting the data, Flipkart might need to clean up inconsistencies in product names, normalize pricing data (e.g., converting all prices to INR if different currencies are used), and aggregate sales data by product category and region. They may also enrich the data by linking it with customer demographics to analyze buying patterns by age group or location.

Load

It's the final step, where the transformed data is loaded into a target data warehouse or a database for analysis. The data warehouse is a centralized repository that business analysts, data scientists, and other stakeholders can query.

Example: After transformation, Flipkart loads the cleaned and aggregated data into its data warehouse. This allows analysts to run queries and generate reports on sales performance, customer behavior, and product trends during the Big Billion Days sale.

.

Big Data Processing

Now we have understood the process, but still one thing is not clear that how this ETL process is executed. So there are various processing engines like Apache Hive and Spark which are used as processing engines. A processing engine is like a powerful tool that helps make sense of large amounts of data. Imagine you have a

huge pile of orders, like millions of orders waiting to be ingested in big data systems. A processing engine takes that pile of orders and sorts through it, analyzing, organizing, and transforming the data into something useful. It's like a super-fast worker that reads, calculates, and summarizes information so that you can easily understand and use it. In simple terms, a processing engine does the heavy lifting of turning raw data into something meaningful and actionable. There are various processing engine like Spark and Hive which provides easy interface like SQL to write systems which helps you to do write the ETL like workload with ease.

Data Integration

Another question that needs to be answered is when will data get ingested. That's where data pipelines comes into the picture. Data Pipelines are Automated workflows for moving and transforming data between systems (e.g., Apache Airflow). **Apache Airflow** is like a smart scheduler for data tasks. Imagine you have a list of chores to do each day—like washing clothes, cooking, and cleaning. You'd want them done in a specific order and at certain times. Apache Airflow helps with that but for data tasks. It sets up and manages complex workflows, ensuring that tasks are done in the right order and at the right times. For example, it can handle tasks like pulling data from different sources, cleaning it up, and loading it into a database without needing to do it manually. So with something like Apache Airflow, you would just need to define how you want to gather data, what data you want to gather and at what conditions/time/frequency you want to gather this data (may be by using cron expression).

Data Consumers

Now once this data is ingested into systems, its ready to use by analysts. There are various ways in which this data can be used . Some of them are :

- Business Intelligence (BI) Tools: Tools for querying, visualizing, and analyzing data (e.g., Tableau).
- Data Science and Machine Learning: Platforms for building models and performing advanced analytics (e.g., Jupyter Notebooks, Databricks).
- Dashboards: Real-time visual displays of key metrics and trends.
- Reports: Generated summaries and detailed views of data insights.