The Big Data Challenge

- Organizations across sectors leveraging big data for competitive advantage
- Data volumes reaching terabytes and petabytes
- Traditional systems struggling with scale and complexity
- Need for cost-effective, scalable solutions

- **What is Big Data?**
  - Large amounts (terabytes+) of poly-structured data
  - Continuous flow through organizations
  - Includes:
    - Video
    - Text
    - Sensor logs
    - Transactional records

- **Business Value of Big Data**
  - Organizations using analytics are 2x more likely to be top performers
  - Enables:
    - Better assessment
    - Anticipation of customer behavior changes
    - Stronger supply chains
    - Improved marketing campaign effectiveness
    - Enhanced business continuity

Data integration

- Data integration is the process of combining data from various sources into a unified view.

- This consolidated view of data enables easier analysis, management, and utilization in applications such as business intelligence (BI), analytics, and reporting.

Why Data Integration Matters

- Organizations often rely on multiple data sources that are dispersed across different databases, applications, and formats.

- Without integration, data remains disconnected, which limits its usefulness and makes comprehensive analysis difficult.

Data integration is Used for:

- **Improve Data Quality and Consistency**: By bringing data into a centralized structure, inconsistencies and errors can be identified and corrected more easily.
- **Enhance Decision-Making:** With a unified view of data, organizations can gain insights and make decisions based on a full picture rather than fragmented information.
- **Enable Advanced Analytics and BI:** Data integration is crucial for feeding data into analytical models, enabling trend analysis, predictions, and actionable insights.

Approaches to Data Integration

- The process typically involves:
  - **Extracting** data from various sources
  - **Transforming** it to fit a consistent schema and quality standards
  - **Loading** it into a target system such as a data warehouse or data lake.

- The two most common methods are **ETL (Extract, Transform, Load)** and **ELT (Extract, Load, Transform)**, each suited to different needs and systems.
  - **ETL** is ideal for traditional data warehousing where data must be cleaned and standardized before analysis.

  - **ELT** is more suitable for cloud environments where large-scale storage and compute resources are available to process raw data directly within the target system.

Use Cases for Data Integration

- Business Intelligence and Reporting: Aggregating data from finance, operations, sales, etc., to gain a holistic view of performance.
- Customer 360 View: Integrating customer data from various touchpoints (e.g., CRM, social media, purchase history) to provide personalized experiences.
- Data Migration and Synchronization: Moving and synchronizing data between systems, especially during technology upgrades or system consolidations.

ETL (Extract, Transform, Load)

- **Extract**: Data is gathered from different sources, often in various formats, such as databases, files, or web services.
- **Transform**: Data is cleaned, formatted, and transformed into the required schema or structure before loading. This may include operations like filtering, joining, aggregating, and deduplicating.
- **Load**: The processed data is then loaded into the target system, typically a data warehouse.
- **Purpose**: ETL is ideal for ensuring data is consistent, cleansed, and formatted before analysis. It has been widely used for batch processing in data warehousing, where data quality and structure are vital before it reaches the target system.
- **Latency**: ETL generally operates in batch mode, meaning data is processed in predefined chunks at intervals, leading to potential delays in data availability for analysis.
- **Architecture**: ETL processes typically run on separate ETL servers before sending data to the data warehouse. This design is efficient for historical data analysis but may be limited in scalability for real-time requirements.

ELT (Extract, Load, Transform)

- **Extract**: Similar to ETL, data is extracted from various sources, including structured and semi-structured data.
- **Load**: Raw data is immediately loaded into the target system, often a data lake or a cloud-based data warehouse that can handle large-scale storage.
- **Transform**: Once the data is in the target system, transformations are applied on-demand using the processing power of the target system. This includes transformations like filtering, cleansing, and aggregation.
- **Purpose**: ELT takes advantage of the high computational power of modern cloud-based data platforms, allowing for scalable and parallelized transformations directly within the data warehouse. This approach is suitable for handling large volumes of diverse data, especially in real-time or near-real-time scenarios.
- **Latency**: Since raw data is loaded immediately, ELT supports more real-time or near-real-time data availability, making it effective for immediate analytics and machine learning applications.
- **Architecture**: Use the computational resources of the data warehouse for transformations, eliminating the need for separate ETL servers and making it cost-efficient and scalable.

Comparison and Use Cases

Aspect	ETL	ELT
Process	Extract-Transform-Load	Extract-Load-Transform
Transformation	Before loading; processed on a separate server	After loading; processed within the target system
Latency	Higher latency; operates in batches	Lower latency; supports real-time/near-real-time
Scalability	Limited by ETL server resources	Highly scalable, utilizing target system resources
Data Suitability	Structured, consistent data for historical reporting	Diverse data types, real-time analytics, and ML
Use Case	Historical data analysis, traditional data warehousing	Real-time reporting, big data, and cloud analytics

ETL VS ELT

- The main difference is when the transformation step occurs in the process
- Choice between them depends on organizational needs, data schema requirements, transformation complexity, performance, and budget
- ETL (Extract, Transform, Load):
  - Benefits:
    - Convenient and simple
    - Early data masking for compliance (GDPR, HIPAA, CCPA)
    - Widely known and understood
  - Challenges:
    - Requires defining schema in advance
    - May strain performance with complex transformations

ETL VS ELT

- ELT (Extract, Load, Transform):
  - Benefits:
    - Faster data availability
    - Better for smaller datasets with simple transformations
    - Less maintenance (cloud-based)
    - Highly scalable
  - Challenges:
    - Can be more expensive
    - Requires more specialized skills
    - Potential security concerns with sensitive data

Choosing Between ETL and ELT

- **Consider**:
  - Destination schema requirements
  - Transformation complexity and dataset size
  - Environment preferences (integration platform vs. data warehouse)
  - Cost structure (flat-rate vs. consumption-based pricing)
  - Team expertise

- **Best Choice**:
  - No universal answer - depends on specific use case
  - May require proof of concept to determine best approach
  - Should align with organization's specific needs and capabilities

Data Ingestion

- Data ingestion is the process of obtaining and importing data for immediate use or storage in a database or data warehouse
- It's the "Extract" part of both ETL and ELT processes
- Involves moving data from various sources to a destination where it can be stored and analyzed

Types of Data Ingestion

A. By Timing:

- Batch Ingestion
  - Collects and imports data in periodic intervals
  - Example: Daily sales data loaded every night at midnight
  - Best for: Large volumes of data that don't need real-time processing
  - Common use: Financial reports, payroll processing

- Real-time/Stream Ingestion
  - Processes data as it's created/received
  - Example: Social media monitoring, stock market data
  - Best for: Time-sensitive information
  - Common use: Fraud detection, monitoring systems

B. By Source:

- Internal Sources
  - Company databases
  - Application logs
  - CRM systems
  - Employee systems

- External Sources
  - Social media feeds
  - Public APIs
  - Third-party databases
  - Web scraping data

Key Components of Data Ingestion:

- A. Source Systems
  - Databases (SQL, NoSQL)
  - File systems
  - APIs
  - Streaming platforms
  - IoT devices
  - Web services
- B. Processing Layer
  - Data validation
  - Format checking
  - Error handling
  - Rate limiting
  - Queue management
- C. Destination Systems
  - Data warehouses
  - Data lakes
  - Analytics platforms
  - Operational databases

Common Challenges

A. Technical Challenges

- Variable data formats
- Source system availability
- Network reliability
- Data volume management
- Performance bottlenecks

B. Data Quality Issues

- Inconsistent formats
- Missing values
- Duplicate records
- Invalid data
- Encoding problems

Tools and Technologies

A. Popular Ingestion Tools

- Apache Kafka
- Apache NiFi
- AWS Kinesis
- Google Cloud Dataflow
- Azure Data Factory

B. Features to Consider

- Scalability
- Reliability
- Error handling
- Monitoring capabilities
- Integration options
- Security features

Security Considerations:

A. Data Protection

- Encryption in transit
- Access controls
- Audit logging
- Compliance tracking

B. Compliance Requirements

- GDPR
- HIPAA
- CCPA
- Industry-specific regulations

Common Use Cases:

A. Business Intelligence

- Sales data collection
- Customer behavior tracking
- Market analysis
- Performance metrics

B. Operational Analytics

- Log analysis
- System monitoring
- User activity tracking
- Resource utilization

C. Big Data Applications

- IoT data collection
- Social media analysis
- Scientific data gathering
- Sensor data processing

Performance Optimization

A. Strategies

- Parallel processing
- Data compression
- Caching mechanisms
- Load balancing
- Resource allocation

B. Monitoring Metrics

- Throughput
- Latency
- Error rates
- Resource utilization
- Queue lengths

Future Trends

A. Emerging Technologies

- AI-powered data ingestion
- Automated data quality
- Smart scaling
- Real-time processing
- Edge computing integration

B. Industry Developments

- Increased automation
- Better self-service options
- Enhanced security features
- Improved compliance tools
- Greater integration capabilities

Traditional ETL vs Hadoop ETL

- They are designed for different data environments and have distinct characteristics.

- In the traditional ETL process, data is extracted from source systems (like databases, APIs, and flat files), transformed to fit a target schema or business rules, and then loaded into a destination system, typically a data warehouse.

- **Key characteristics include:**
- **Batch Processing:** Traditional ETL is often a batch process, running at set times (e.g., nightly) to update data in the warehouse.
- **Relational Databases**: The destination is usually a relational database (like SQL Server, Oracle, or MySQL), where structured data is organized in a tabular format.
- **Data Volume:** Designed for structured data with limited volume; not ideal for massive, unstructured, or semi-structured data.
- **Scalability:** Scaling is limited, as relational databases and ETL tools can become slow and expensive when handling massive data volumes.
- **ETL Tools:** Popular tools include Informatica, Talend, and Microsoft SSIS, which are powerful for traditional databases but may not handle Big Data efficiently.

Hadoop ETL

- Hadoop ETL is optimized for processing Big Data in distributed environments using technologies in the Hadoop ecosystem, such as HDFS (Hadoop Distributed File System), Hive, and Spark.

- **Key characteristics include:**
- **Distributed Processing**: Hadoop uses a cluster of machines, enabling distributed data storage and parallel processing, making it ideal for handling massive data volumes.
- **Variety of Data:** Hadoop ETL can handle structured, semi-structured, and unstructured data types (e.g., text files, logs, images) through tools like Hive, HBase, and Spark.
- **Real-Time or Near-Real-Time:** While traditional Hadoop MapReduce is batch-oriented, modern frameworks (e.g., Spark) provide real-time or near-real-time data processing.
- **Scalability:** Hadoop scales horizontally by adding more nodes, which is cost-effective for growing data volumes.
- **ETL Tools and Technologies**: ETL in Hadoop may use tools like Apache Nifi, Talend, or custom workflows using Spark and Hive to perform transformations on large datasets directly in the Hadoop cluster.

Traditional ETL vs Hadoop ETL

Traditional ETL:

- Fixed schemas
- Limited scalability
- High infrastructure costs
- Complex transformations

Hadoop ETL:

- Schema-on-read
- Highly scalable
- Cost-effective
- Flexible processing

Traditional ETL vs Hadoop ETL

Benefits of Hadoop for ETL

- No schema requirements on data ingestion
- Massive parallel processing
- Cost-effective storage
- Flexible data formats
- Scalable performance

ETL Patterns with Hadoop

- Traditional ETL
- ELT (Extract, Load, Transform)
- ETLT (Extract, Transform, Load, Transform)
- Hybrid approaches

Data Ingestion and Data Transformation

Data Ingestion with Hadoop

- Multiple source support
- Raw data storage
- Parallel ingestion
- Real-time capabilities
- Batch processing options

Data Transformation in Hadoop

- Distributed processing
- Complex transformations
- Data quality checks
- Data enrichment
- Aggregations

HDFS (Hadoop Distributed File System)

- Distributed file system designed for reliable, scalable storage
- Built to handle very large files across commodity hardware
- Follows write-once, read-many access model

Key Components

NameNode

- Master server managing filesystem namespace
- Maintains metadata and file-to-block mapping
- Tracks file system health and replication
- Single point of coordination

DataNodes

- Slave nodes storing actual data blocks
- Handle read/write requests from clients
- Send heartbeats to NameNode
- Manage block replication

HDFS (Hadoop Distributed File System)

Key Features

1. Block Storage
   - Files split into fixed-size blocks (default 128MB)
   - Each block replicated across multiple DataNodes
   - Replication factor typically set to 3

2. Fault Tolerance
   - Automatic block replication
   - Secondary NameNode for metadata backup
   - Automatic recovery from DataNode failures

3. Scalability
   - Supports thousands of nodes
   - Handles petabytes of data
   - Linear scalability with additional nodes

HBase (Distributed NoSQL Database)

Overview
- Column-oriented, distributed NoSQL database
- Built on top of HDFS
- Provides real-time, random access to big data

Architecture Components

1. HMaster
- Manages and coordinates Region Servers
- Handles schema changes
- Manages load balancing

2. **Region Servers**
- Handle data storage and retrieval
- Manage regions (ranges of rows)
- Process read/write requests

3. **ZooKeeper**
- Coordinates distributed operations
- Maintains cluster state
- Provides failover support

HBase (Distributed NoSQL Database)

Data Model

1. Tables
   - Divided into regions
   - Distributed across Region Servers

2. Row Keys
   - Unique identifiers
   - Basis for data distribution

3. Column Families
   - Groups of related columns
   - Flexible schema design

Use Cases

- Real-time random read/write access
- Storing sparse data
- Time-series data
- Online transaction processing

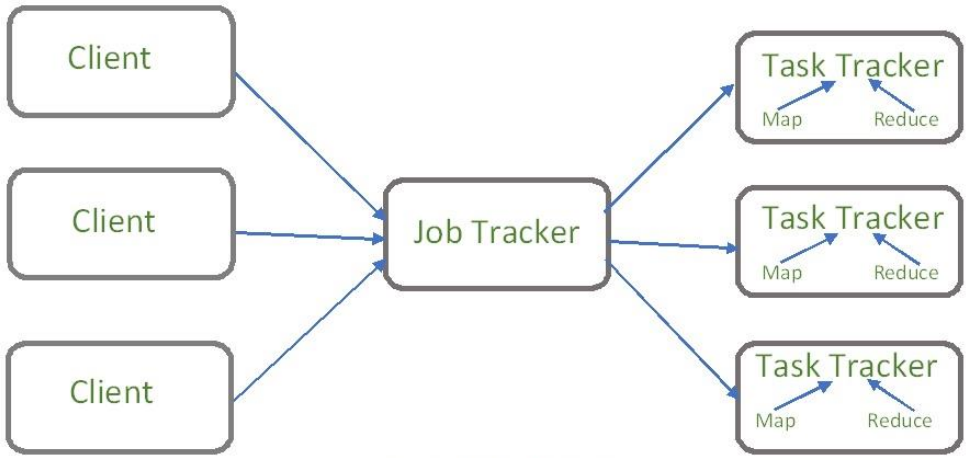
YARN (Yet Another Resource Negotiator)

What is YARN?
- Cluster resource management system introduced in Hadoop 2.0
- Separates resource management from processing logic
- Called the "operating system" of Hadoop

Evolution and History
- Introduced to overcome MapReduce V1 limitations
- Enables multiple processing frameworks
- Supports both batch and stream processing

YARN (Yet Another Resource Negotiator)

- YARN stands for “Yet Another Resource Negotiator“.
- It was introduced in Hadoop 2.0 to remove the bottleneck on Job Tracker which was present in Hadoop 1.0.
- YARN was described as a “Redesigned Resource Manager” at the time of its launching, but it has now evolved to be known as large-scale distributed operating system used for Big Data processing.
- YARN architecture basically separates resource management layer from the processing layer.



Hadoop 1.0 architecture

- In Hadoop 1.0 version, the responsibility of Job tracker is split between the resource manager and application manager.

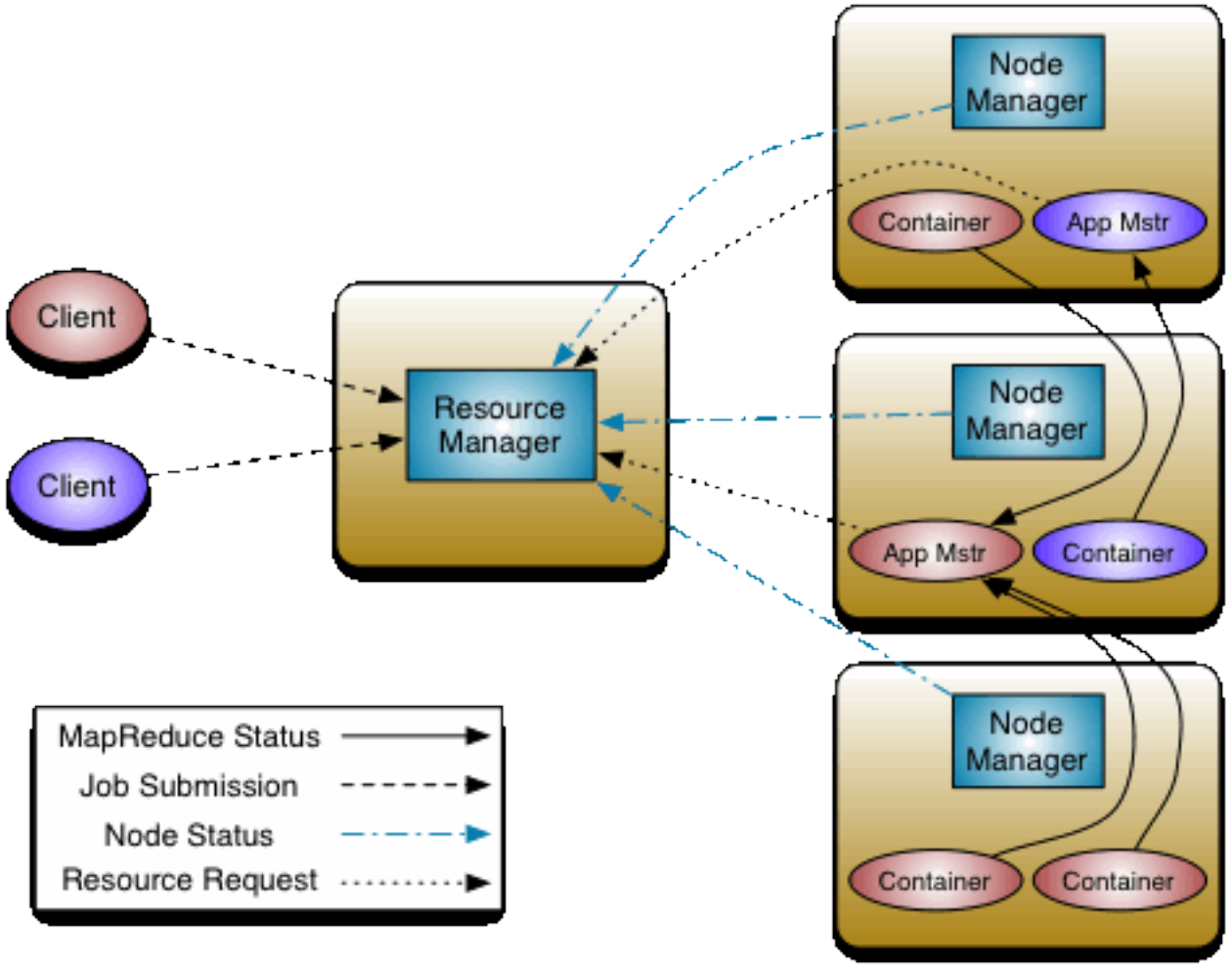
YARN (Yet Another Resource Negotiator)

Key Benefits

1. Multi-tenancy
   - Multiple frameworks run simultaneously
   - Efficient resource sharing
   - Better cluster utilization

2. Scalability
   - Supports thousands of nodes
   - Handles petabytes of data
   - Dynamic resource allocation

3. Flexibility
   - Framework-agnostic
   - Supports diverse workloads
   - Pluggable scheduler architecture



YARN Workflow Overview

- **Job Submission**
  - Client submits a job, initiating the ApplicationMaster for resource requests.
- **Resource Allocation**
  - ResourceManager assigns resources based on available capacity and policies.
- **Execution and Monitoring**
  - NodeManager launches containers, ApplicationMaster monitors, and manages tasks.
- **Job Completion**
  - On completion, ApplicationMaster releases resources back to the cluster.

Core Components of YARN

- **ResourceManager**
  - Central authority for managing cluster resources.
  - Two key components:
    - **Scheduler**: Allocates resources based on policy.
    - **Application Manager**: Manages applications' resource requests and monitoring.

- **NodeManager**
  - Runs on each cluster node to monitor resource usage and manage application containers.

- **ApplicationMaster**
  - Manages the lifecycle of applications (per job or per application).
  - Responsible for negotiating resources with ResourceManager and working with NodeManager.

Resource Manager (RM)

Resource Manager (RM)

1. Scheduler
   - Allocates resources to applications
   - Implements scheduling policies
   - Manages resource queues
   - Tracks resource availability

2. Applications Manager
   - Accepts job submissions
   - Manages Application Master lifecycle
   - Coordinates with Node Managers
   - Handles failover scenarios

Resource Management

Resource Types

1. Memory
   - Allocated in MB/GB
   - Configurable limits
   - Monitored continuously

2. CPU
   - Allocated in virtual cores
   - CPU scheduling
   - Processing power management

3. Other Resources
   - Disk space
   - Network bandwidth
   - GPU allocation (with extensions)

Resource Allocation Process

1. Request Phase
   - Application submits requirements
   - Resource Manager evaluates availability
   - Queue placement determined

2. Allocation Phase
   - Resources reserved
   - Containers allocated
   - Node Manager notified

3. Monitoring Phase
   - Usage tracking
   - Resource adjustment
   - Preemption if necessary

Node Manager (NM)

1. Resource Management
   - Manages node-level resources
   - Reports to Resource Manager
   - Monitors resource usage
   - Enforces resource limits

2. Container Management
   - Creates and destroys containers
   - Monitors container health
   - Reports container status
   - Manages local resources

Application Master (AM)

1. Application Lifecycle
   - Negotiates resources
   - Tracks application progress
   - Handles failures
   - Coordinates task execution

2. Resource Negotiation
   - Requests containers from RM
   - Allocates tasks to containers
   - Monitors resource usage
   - Releases unused resources

Scheduling in YARN

Capacity Scheduler

1. Features
   - Hierarchical queues
   - Resource guarantees
   - Elastic allocation
   - Multi-tenancy support

2. Configuration
   - Queue definitions
   - Resource limits
   - Access control
   - Preemption settings

Fair Scheduler

1. Features
   - Dynamic resource pools
   - Fair sharing
   - Resource preemption
   - Queue weights

2. Policies
   - FIFO
   - Fair sharing
   - DRF (Dominant Resource Fairness)

Security and Access Control

Authentication

1. Kerberos Integration
   - Secure authentication
   - Token-based delegation
   - Service principal management

2. Service Level Authentication
   - Inter-service security
   - Client authentication
   - Token management

Authorization

1. Queue Access Control
   - User/group permissions
   - Queue placement rules
   - Resource limits

2. Application Level Security
   - Container isolation
   - Resource access control
   - User impersonation

Key Features of YARN

- **Scalability**
  - Enables Hadoop to scale to thousands of nodes, handling large data volumes.
- **Flexibility**
  - Supports a variety of data processing frameworks (MapReduce, Spark, etc.).
- **Resource Utilization**
  - Optimizes the use of cluster resources for multiple applications running simultaneously.
- **Fault Tolerance**
  - Provides resilience through monitoring, job restarts, and resource reallocation.

Benefits of YARN in Hadoop Ecosystem

- **Enhanced Multitenancy**
  - Multiple users and applications can run simultaneously on the same cluster.
- **Better Cluster Utilization**
  - Efficiently allocates resources, reducing idle times.
- **Improved Performance**
  - Parallel processing and dynamic resource allocation increase throughput.