BOSSCODER ACADEMY

DATA ENGINEERING ROADMAP

DATA ENGINEERING LIFECYCLE

DATA WAREHOUSING

PYTHON & LIBRARIES

ORCHESTRATION TOOLS

Your Ultimate Roadmap To Becoming A Data Engineer

Disclaimer

EVERYONE LEARNS UNIQUELY.

This roadmap is a guide to help you navigate the journey of becoming a Data Engineer.

Treat this as direction, not a fixed rulebook. Real growth comes from consistent practice and solving real-world problems.

Introduction —

WHY FOLLOW THIS ROADMAP?

- Focus on the right way to become a Data Engineer — a clear, structured path that avoids confusion and wasted effort.
- Designed with tools and practices used by FAANG-level and top tech companies.
- Includes hands-on projects for practical, job-ready experience.
- Prepares you for interviews and real-world, large-scale data engineering challenges.

TABLE OF CONTENTS

PHASE 1:

Data Engineer Basics :

01 Who Is A Data Engineer?

A Data Engineer is a professional who designs and maintains the systems that allow data to be collected, processed, and stored for business use. They make sure data flows smoothly from different sources into a usable format for analysis.

THINK OF IT THIS WAY:

- Data is like **water in nature** - raw and scattered.
- A Data Engineer builds the **pipes and filtration systems** that clean and deliver this water to homes.
- Similarly, they build **data pipelines** and **storage systems** that make raw data accessible, reliable, and structured.

Their role is essential for businesses because without them, data scientists and analysts would struggle to get the clean, organized data they need.

02 Why Data Engineering?

Companies generate massive amounts of data every day from apps, websites, transactions, and sensors. But this raw data is:

- **MESSY** – full of errors and duplicates.

- **SCATTERED** – stored across multiple systems.
- **UNUSABLE** – without structure or context.

Data Engineers turn this into structured, usable information.

For Example: Amazon collects data from many places, not just users. Every part of its business generates information that needs to be captured and organized:

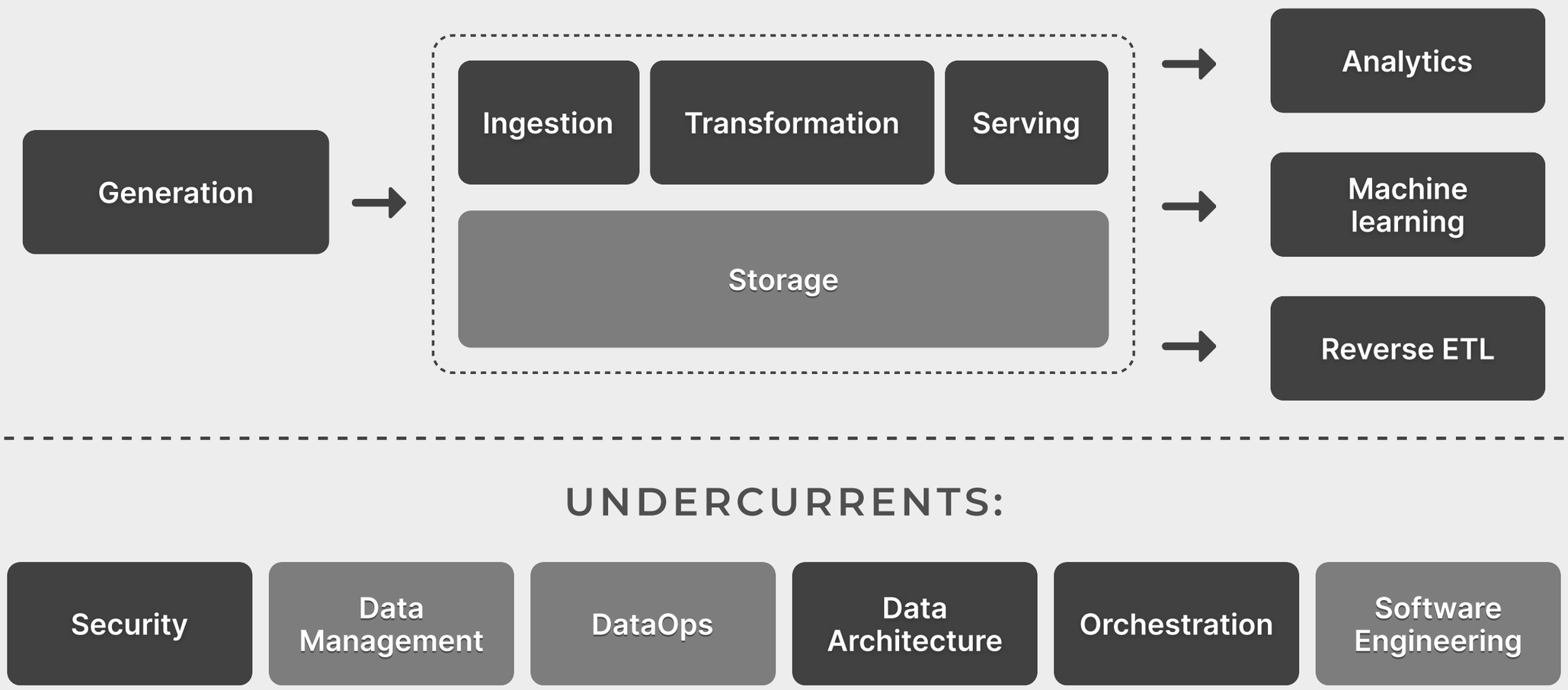
- User Data – clicks, searches, purchases, reviews.
- Shipping Data – delivery times, package tracking.
- Warehouse Data – stock levels, product movement.
- Seller Data – prices, product listings, seller ratings.

WHAT DATA ENGINEERS DO?

- Collect all this data from different systems.
- Clean it (remove errors, duplicates, messy formats).
- Store it in data warehouses (like Redshift or Snowflake).
- Make it ready for use by other teams.

03 Data Engineering Lifecycle

The Data Engineering Lifecycle is the story of data from start to finish.
It begins the moment data is **created**, whether from a user clicking a button, a sensor sending a reading, or a server generating logs — and ends when that data is finally **used for decision-making, analytics, or machine learning models.**



Data Generation

This is where raw data originates. It can come from:

- Applications (e.g., e-commerce transactions)
- Sensors and IoT devices or Logs from the Servers

For Example: This is where raw data is created.

- **Netflix:** Every time you play, pause, or rate a movie.
- **Amazon:** When you search for a product or add it to your cart.
- **Uber:** Each ride request, driver location update, or payment.

Ingestion

Data needs to be brought into a system from multiple sources. Ingestion can be:

- **Batch-based** (data collected at intervals)
- **Streaming** (real-time data flow)

For Example: Getting that data into a central system.

- **Netflix:** Collects millions of “watch events” per second and streams them into Kafka.
- **Amazon:** Batch uploads sales data every hour + streams real-time click data.
- **Uber:** Streams live driver GPS locations into systems for real-time tracking.

Storage

Once ingested, data must be stored efficiently so it can be processed later. Depending on its nature:

- **Structured data:** stored in **data warehouses** like Snowflake or BigQuery.
- **Unstructured/semi-structured data:** stored in **data lakes** like S3 or HDFS.

Transformation

Raw data is rarely clean or ready for use. Transformation involves:

- Cleaning (removing duplicates, fixing errors)
- Normalizing and aggregating data
- Formatting for downstream applications

For Example: Cleaning and organizing messy data into usable formats.

- **Netflix:** Converts raw clickstream logs into session-level summaries (one row per user session).
- **Amazon:** Cleans inventory data so “iPhone 14” and “iPhone-14” aren’t treated as two separate products.
- **Uber:** Combines driver GPS logs into a single trip record (start point, route, end point).

Serving

After processing, data is served for use:

- Business Intelligence dashboards
- Machine Learning models
- APIs for external systems

For Example: Making processed data available to others.

- **Netflix:** Feeds structured watch-history into ML models for personalized recommendations.
- **Amazon:** Serves sales and user behavior data to dashboards for product managers.
- **Uber:** Provides APIs so the app can show live driver ETAs to riders.

Consumption

This is where business value is created:

- **Analytics:** Dashboards for decision-making
- **Machine Learning:** Training recommendation systems
- **Reverse ETL:** Sending processed data back into business tools like CRMs

Undercurrents

Throughout the lifecycle, these elements run across every stage:

- **Security:** Protecting sensitive data
- **Data Management:** Maintaining quality and governance
- **DataOps:** Automating workflows for reliability
- **Architecture & Orchestration:** Designing scalable systems
- **Software Engineering:** Building robust tools and pipelines

Ajay Naik

ansrsource Learn Toward Tomorrow™ → amazon

My experience with Bosscoder was ***life-changing.*** The program gave me v***aluable mentorship, resume optimization, and structured assignments.*** The teaching was excellent, and the supportive team boosted my confidence. It turned out to be a ***great investment for career growth.***

PHASE 2:

Core Fundamentals

WHY CORE FUNDAMENTALS MATTER?

Before exploring Big Data tools, cloud platforms, or streaming systems, it's essential to understand **how data is stored and retrieved efficiently.**

- **DBMS (Database Management System)** to store and manage data.
- **SQL (Structured Query Language)** to query and manipulate data.
- **Python and its Libraries** – to process, clean, and automate data pipelines.

01 What Is DBMS?

A Database Management System (DBMS) is software that enables users and applications to **store, organize, and retrieve data efficiently**. It acts as an interface between the user and the database.

WHY DO WE NEED DBMS?

- To avoid **data redundancy** (duplicate data).
- To ensure **data integrity** and **consistency.**
- To support **multiple users concurrently** without conflicts.
- To enforce **security** and control access.

TYPES OF DATABASES:

1. **RDBMS (Relational Database Management System):**

- Organizes data in tables (rows & columns).
- Supports relationships between tables.
- Examples: MySQL, PostgreSQL, Oracle.

2. **NoSQL Databases:**

- For unstructured or semi-structured data.
- Types: Document (MongoDB), Key-Value (Redis), Columnar (Cassandra), Graph (Neo4j).
- Common in **real-time applications.**

Core DBMS Concepts

This is where business value is created:

- **Primary Key:** Uniquely identifies a record in a table.
- **Foreign Key:** Creates a link between two tables.
- **Normalization:**
    a. **1NF:** Remove repeating groups.
    b. **2NF:** Remove partial dependency.
    c. **3NF:** Remove transitive dependency.
- **Indexes:** Improve query performance by reducing scan time.
- **Transactions & ACID Properties:**
    a. **Atomicity:** All or nothing.
    b. **Consistency:** Maintains integrity.
    c. **Isolation:** Prevents transaction interference.
    d. **Durability:** Ensures changes persist after commit.

WHY NORMALIZATION?

- Reduces redundancy and anomalies (update, insert, delete).
- Makes data **consistent and organized.**
- However, in **analytics systems**, denormalization is common for speed.

02 SQL – Structured Query Language

SQL is a standard programming language used to interact with Relational Database Management Systems (RDBMS). It is the foundation of working with structured data because it allows us to define how data is stored, retrieve information when needed, and manage the data efficiently.

CORE COMPONENTS OF SQL

- **DDL (Data Definition Language):** Commands to define schema (CREATE, ALTER, DROP).
- **DML (Data Manipulation Language):** Commands to manage data (INSERT, UPDATE, DELETE).
- **DQL (Data Query Language):** Command to query data (SELECT).
- **DCL (Data Control Language):** Permissions (GRANT, REVOKE).
- **TCL (Transaction Control Language):** COMMIT, ROLLBACK, SAVEPOINT.

BASIC SQL OPERATIONS

- **Purpose:** Fetch data from one or more tables. SELECT * FROM customers
- **WHERE – Filtering Data:** SELECT name, city FROM customers
  WHERE city = 'Delhi'
- **JOIN – Combine Data from Multiple Tables:** SELECT c.name, o.amount
  FROM customers c
  INNER JOIN orders o ON c.customer_id = o.customer_id;

- **Subquery – Query Inside a Query:**
  SELECT name FROM customers
  WHERE customer_id IN (
  SELECT customer_id FROM orders WHERE amount > 1000);

ADVANCED SQL TOPICS:

Once you are comfortable with the basics of SQL (SELECT, JOIN, WHERE, Subqueries), the next step is to learn Advanced SQL. These concepts are important because real-world data engineering problems involve large datasets, complex analytics, and performance optimization.

1. **Window Functions (Analytical Functions)**
   - Used for calculations across sets of rows without collapsing them.
   - Common functions: ROW_NUMBER(), RANK(), DENSE_RANK(), LAG(), LEAD(), NTILE().
2. **CTEs (Common Table Expressions) & Recursive Queries**
   - CTEs make queries more readable and reusable. Recursive queries help with hierarchical data.
   - Useful for **organization charts, tree structures, graph traversal.**
3. **Advanced Joins**
   - **Self Joins** (comparing a table with itself).
   - **Cross Joins** (Cartesian product).
   - **Full Outer Joins** (when you need all rows from both tables).
4. **JSON & Semi-Structured Dat**
   - Modern warehouses (Postgres, BigQuery, Snowflake) allow querying JSON directly.

5. **Set Operations:** UNION, INTERSECT, EXCEPT.

> You can also refer to the **“SQL Roadmap from Zero to Advance”** by Bosscoder Academy to follow a structured path, get hands-on practice, and strengthen your SQL & database skills.

03 Python & Its Libraries

Python is the heart of Data Engineering. Almost every pipeline, automation, or data workflow uses it. The reason is simple: Python is easy to learn, has a clean syntax (it reads like English), and comes with powerful libraries built for handling data.

Python Fundamentals

Before using libraries, you must first learn the core parts of the language:

- **Data Types:** Python can handle numbers (int, float), text (strings), and collections like lists, tuples, sets, and dictionaries. These are just different ways to organize data.
- **Variables:** Used to store data values. For example, x = 100 stores a number in variable x.
- **Operators:** Perform actions like addition, subtraction, or comparison.
- **Control Flow:** With if-else and loops, you can control the logic of your program.
- **Functions:** Small reusable blocks of code. Instead of writing the same logic again and again, you put it in a function and just call it.
- **Error Handling:** Using try-except to handle problems without breaking the program.

- **Object-Oriented Programming (OOP):** Concepts like classes, objects, inheritance, and encapsulation help you organize code for bigger projects.

For Example: Imagine Amazon collects daily sales data from different warehouses. Sometimes, the file might have missing values in the “quantity” column. A Python script using Pandas can automatically scan the dataset, fill the missing quantities with 0 (or an average), and then load the cleaned data into a database.

Python Libraries For Data Engineering

Once you know the basics, you unlock the real power of Python: libraries. These are pre-built tools that save you time and effort.

1. **NumPy**
   - Best for numbers, arrays, and mathematical operations.
   - Handles huge datasets much faster than normal Python lists.
   - Example: Calculating the average delivery time for 1 million Amazon orders.
2. **Pandas**
   - The most important library for working with tabular data (rows and columns).
   - Lets you clean, filter, group, and join datasets easily.
   - Example: Cleaning an e-commerce dataset by removing duplicate orders and filling missing values.
3. **Matplotlib & Seaborn**
   - Used for data visualization.
   - Matplotlib is great for basic charts, while Seaborn is built on top of it for advanced and beautiful plots.
   - Example: Plotting a graph of Netflix’s daily watch hours to spot viewing trends.

4. **PySpark**
    - A Python API for Apache Spark, used when working with big data.
    - Lets you process massive datasets across multiple machines.
    - Example: Processing billions of Uber ride logs to calculate surge pricing.
5. **Other Useful Libraries**
    - SQLAlchemy: Run SQL queries directly from Python.
    - Requests: Collect data from APIs (like weather or stock data).
    - Boto3 / Google Cloud SDK: Work with AWS S3, GCP, and other cloud storage systems.
    - Airflow / Prefect SDKs: Automate and schedule data pipelines.

Why Python Is Essential For Data Engineers

- **Flexibility:** You can use Python for everything from small scripts to massive data pipelines.
- **Integration:** Python connects easily with SQL, cloud platforms, and big data tools.
- **Efficiency:** With libraries, you can process millions of records in just a few lines of code.
- **Industry Standard:** Almost every company (Netflix, Amazon, Uber, etc.) uses Python in their data systems.

PHASE 3:

Data Engineering Essentials

This phase covers the core concepts every Data Engineer must know before moving into advanced systems. These concepts explain how data is stored, modeled, and analyzed at scale.

01 What Is OLTP And OLAP?

OLTP (ONLINE TRANSACTION PROCESSING)

- Handles day-to-day business transactions.
- Designed for speed and accuracy.
- Focuses on inserting, updating, or deleting records quickly.
- Supports real-time operations.

For Example: On Amazon, when a customer places an order, the system instantly records the transaction with details like product ID, payment, and shipping address. This is OLTP in action — fast, real-time processing.

OLAP (ONLINE ANALYTICAL PROCESSING)

- Designed for analysis and reporting.
- Works with large volumes of historical data.
- Focuses on complex queries and aggregations.

For Example: On Amazon, analysts use OLAP queries on a data warehouse to generate a quarterly sales report, comparing revenue across categories, regions, and customer segments. This helps decide which products to promote or restock.

OLAP IN DATA ENGINEERING

For Data Engineers, OLAP is where the real value of data comes to life. While OLTP systems capture raw transactions (like sales or rides), OLAP systems allow businesses to:

- Analyze historical data for patterns and trends.
- Power dashboards for business intelligence tools like Power BI, Tableau, or Looker.
- Support decision-making by running complex queries on data warehouses.
- Feed ML models that rely on aggregated, structured datasets.

02 What Is Data Modeling?

Data Modeling is the process of designing how data is stored, connected, and organized in a database or data warehouse. It acts as a blueprint for how data will be structured and accessed.

GOALS OF DATA MODELING:

- Organize data efficiently.
- Avoid redundancy and maintain integrity.
- Optimize for queries and analytics.

LEVELS OF DATA MODELING:

1. **Conceptual Model:**
    - High-level representation of entities and relationships
    - Focuses on what data is important, not how it’s stored.
    - Example: Customers purchase Products.

2. **Logical Model**
    - Adds details like attributes, keys, and constraints (but no physical implementation yet).
    - Example: Customer table with attributes: customer_id, name, city.
3. **Physical Model:**
    - Actual implementation in a DB system (PostgreSQL, Snowflake).
    - Includes indexes, partitions, data types.

> Each level builds on the previous, moving from idea → design → implementation.

TYPES OF DATA MODELS

1. **Relational Model**
    - Organizes data into tables (relations).
    - Uses Primary Keys and Foreign Keys to define relationships.
    - Standard for OLTP systems (transactions) and many OLAP systems (analytics).
    - Example: In Amazon's OLTP database, a Customer table is linked to an Orders table using customer_id.
2. **Dimensional Model**
    - Common in data warehouses for analytics.
    - Organizes data into Facts (measurable events) and Dimensions (descriptive attributes).
    - Example : In Netflix's warehouse, a Fact table stores watch events, while Dimension tables store user details, movie details, and time.

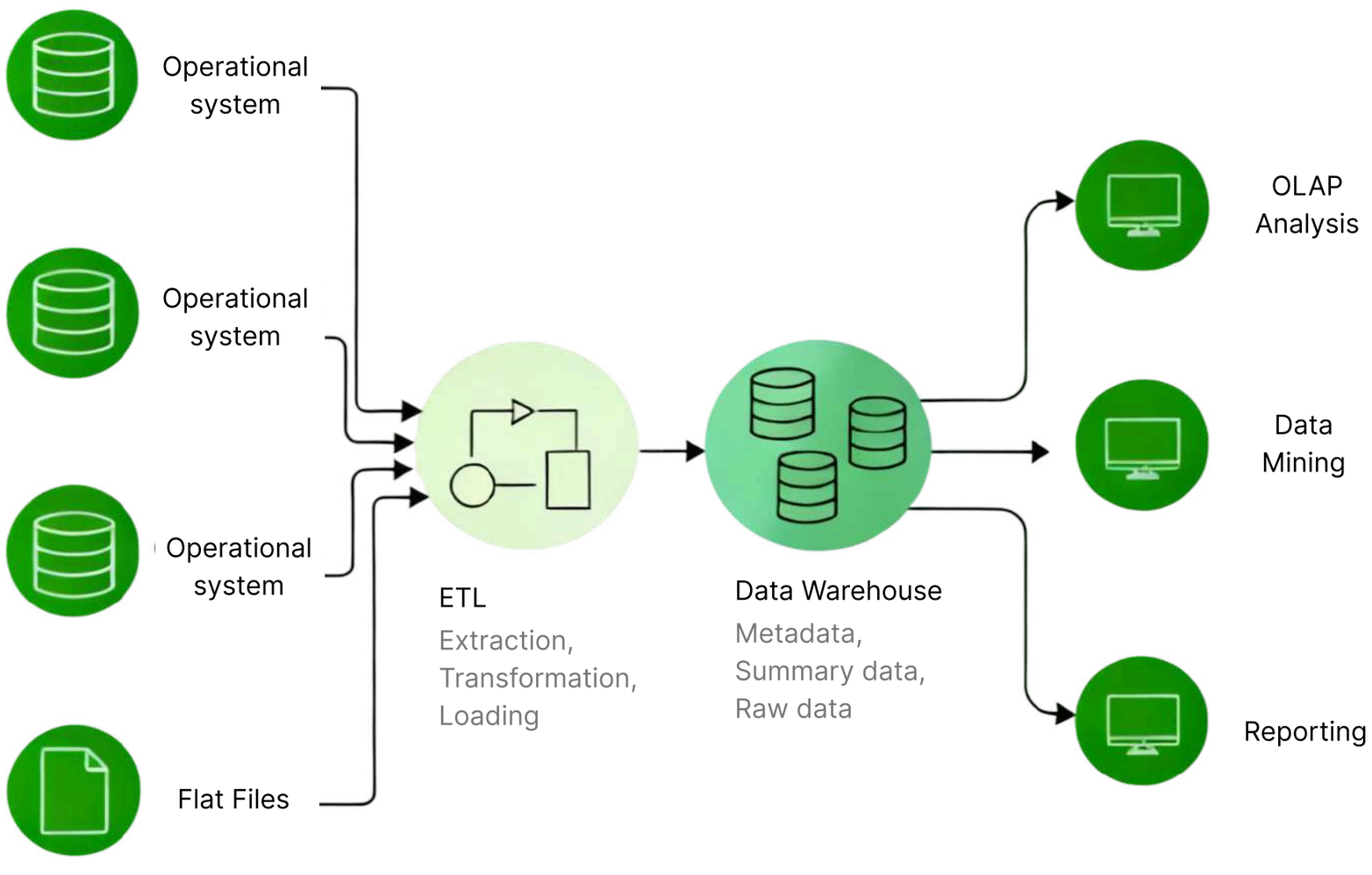
03 What Is A Data Warehouse?

A Data Warehouse is a central place to store data from different sources like apps, websites, CRM, or logs. Instead of handling daily transactions, it is built for analytics, reporting, and decision-making by combining and organizing historical data in one place.

THINK OF IT LIKE A GIANT LIBRARY OF DATA

- Every book (dataset) comes from a different place.
- Before putting it on the shelf, it is cleaned, labeled, and organized.
- Once stored, anyone (analysts, data scientists, business teams) can quickly find and use the information they need.

For Example: At Amazon, sales data from different countries, customer data from CRM, and delivery data from logistics systems are all combined into a warehouse. Analysts then run queries like **"Which region had the highest sales last quarter?" or "Which product categories are growing fastest?"**



TYPES OF DATA WAREHOUSES:

1. **On-Premise Data Warehouses**

- Hosted on a company's own servers.
- **Examples:** Teradata, Oracle, IBM Netezza.
- **Pros:** Full control, customizable security, suitable for strict compliance needs.
- **Cons:** Expensive to scale, requires hardware and maintenance.

2. **Cloud Data Warehouses**

- Fully managed, scalable, and cost-effective.
- **Examples:** Snowflake, Google BigQuery, Amazon Redshift, Azure Synapse.
- **Pros:** Pay-as-you-go, elastic scaling, integrates well with cloud ecosystems.
- **Cons:** Ongoing cost management is needed; less control compared to on-prem.

3. **Hybrid Data Warehouses**

- Combine on-premise + cloud storage.
- Useful when migrating to the cloud or keeping sensitive data on-prem.
- **Examples:** Oracle Exadata Cloud at Customer, IBM Hybrid Data Warehouse.

PHASE 4:

Data Engineering Tools

After understanding **data modeling, SQL, and warehousing,** the next step is learning tools that handle data at scale. These tools enable **big data processing, real-time streaming, and workflow orchestration**—skills every modern Data Engineer must master.

01 Big Data Fundamentals

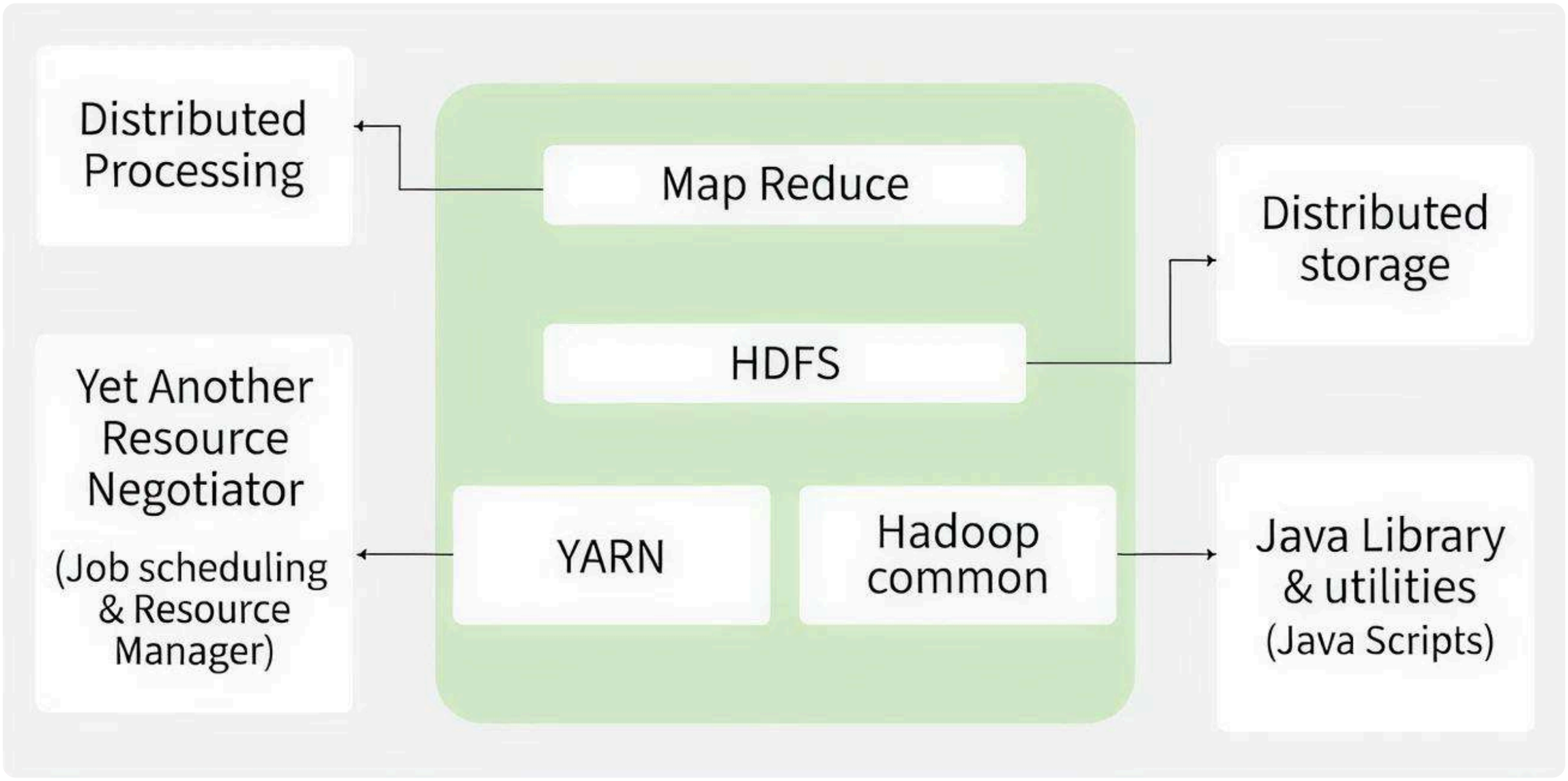
WHAT IS BIG DATA?

- Data that is too large for traditional systems to process efficiently.
- Defined by **3 Vs:**
    a. **Volume:** Large datasets (TBs, PBs).
    b. **Velocity:** Rapidly generated (real-time logs, IoT data).
    c. **Variety:** Structured, semi-structured, and unstructured data.

Hadoop

Hadoop is a tool that helps store and process **huge amounts of data** by using **many computers working together.**

- It splits big data into smaller parts and stores them across different machines.
- Then, it processes those parts in parallel, which makes handling massive data much easier.
- It is mostly used for **batch processing (working on large datasets at once).**



KEY COMPONENTS

1. **HDFS (Hadoop Distributed File System)**
    - A **distributed file system** designed to store very large datasets reliably across clusters.
    - Splits files into **blocks** (128 MB or 256 MB) and replicates them (default 3 copies) for **fault tolerance.**
    - **Main roles:**
        a. **NameNode (Master): Stores metadata (file names, block locations).**
        b. **DataNodes (Workers): Store actual data blocks.**
    - **Why Important:** Ensures high availability and scalability.

2. **MapReduce – Batch Processing Framework**
    - A programming model for processing big data in parallel.

- Two main steps:
    a. **Map():** Splits data and processes chunks locally on DataNodes.
    b. **Reduce():** Aggregates intermediate results into final output.
- **Why Important:** Enables **parallelism** and **fault-tolerant computation.**

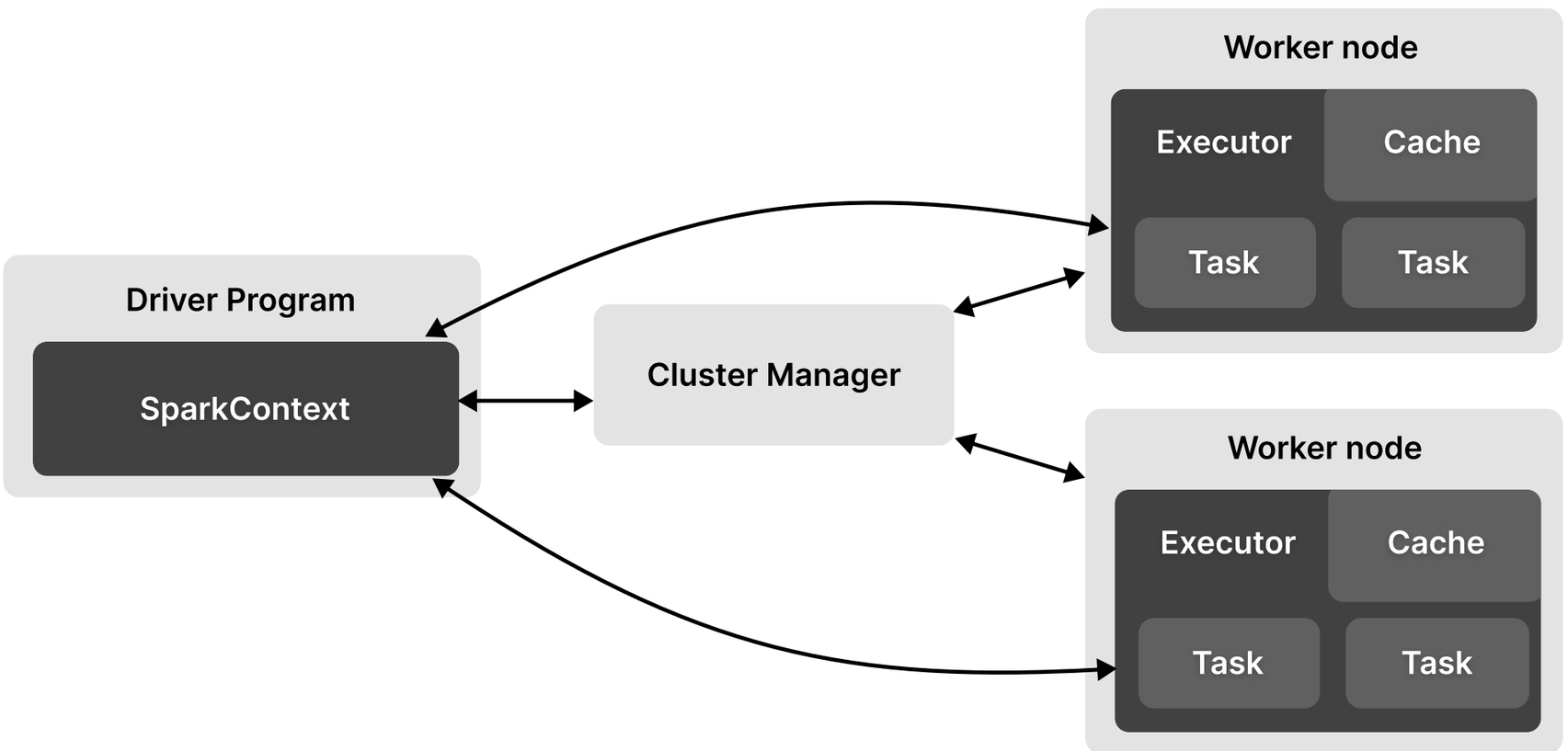
HOW THEY WORK TOGETHER?

HDFS stores the data → MapReduce processes the data → Output is written back to HDFS.

Apache Spark

Apache Spark is a tool used to process very large amounts of data quickly. Instead of running on a single computer, it works on many computers at the same time (distributed processing).

- It is fast because it can keep data in memory (RAM) instead of reading from disk again and again.
- It can handle both batch data (big files, reports) and streaming data (real-time logs, live events).
- **Supports:**
    a. **DataFrame API for easy data transformations.**
    b. **Streaming data processing for real-time workloads.**
    c. **Machine Learning (MLlib) and Graph Processing (GraphX).**
- Works with **multiple languages**: Python (PySpark), Scala, Java, R.
- Can run on **standalone clusters** or with **Hadoop/YARN** and cloud environments.



For Example: Netflix uses Spark to process millions of “watch events” (play, pause, stop) in real time, so it1 can update recommendations almost instantly.

PySpark (Python API For Apache Spark)

Python-first way to build scalable, distributed data pipelines on top of Apache Spark. Great for batch + streaming workloads, ETL/ELT, ML feature engineering, and interactive analytics.

WHY USE PYSPARK

- **Scale-out processing:** Handle TB–PB datasets across clusters without changing your code.
- **Unified engine:** Batch (Spark SQL, DataFrame), streaming (Structured Streaming), ML (MLlib), graph (GraphFrames), and SQL—all in one runtime.
- **Pythonic productivity:** Use Python libraries while leveraging JVM-backed performance.

USE CASES OF PYSPARK

- **Processing Clickstream Logs:** Handle terabytes of click events from millions of users to identify trending products, session paths, and conversion funnels.
- **ETL / Data Lake Pipelines:** Ingest raw data (CSV/JSON/Avro/Parquet) from cloud storage, clean and transform it into curated parquet/Delta tables for analytics.
- **Real-Time Analytics:** Use Structured Streaming with Kafka or Kinesis to power dashboards for live product trends, ad performance, or fraud detection.

Ritesh Chikatwar

IBM → mastercard

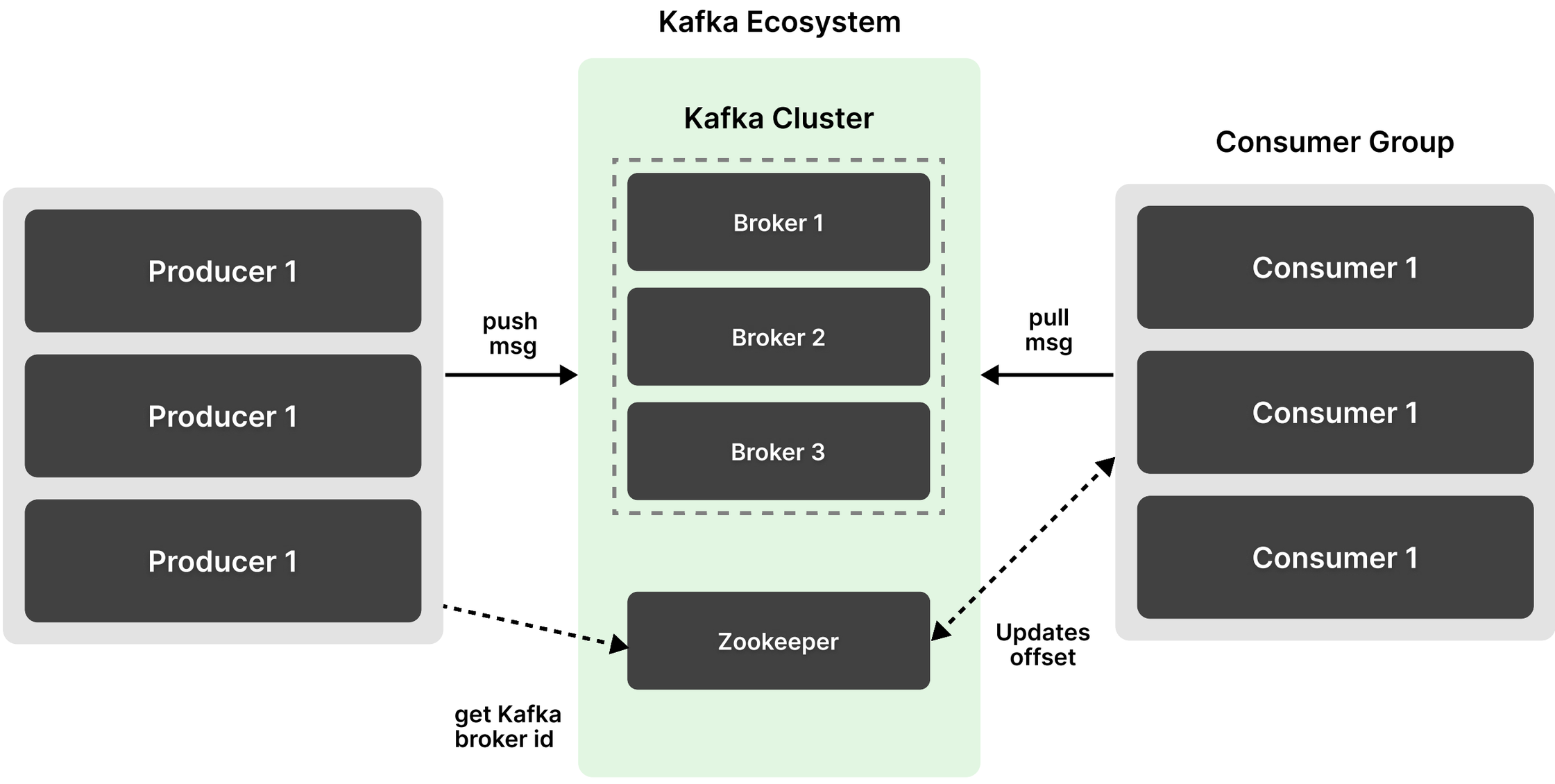
I joined Bosscoder when I was struggling to switch roles after years of experience. The structured curriculum and interactive sessions helped me build clarity. ***One-on-one mentorship and consistent feedback*** kept me on track. Placement support also improved my professional profile. ***I'm thankful to Bosscoder*** for this smooth transition.

02 Distributed Messaging System – Kafka

Apache Kafka

Apache Kafka is a tool that helps move data from one place to another in real time.

- It works like a messaging system: one system sends data (producer), Kafka holds it for a short time, and another system receives it (consumer).
- Kafka makes sure the data flows smoothly, even when millions of events are happening every second.



KEY COMPONENTS

1. **Producers**
    - Applications that **publish (send) messages** to Kafka.
    - Example: An e-commerce website pushing order events to Kafka.
2. **Kafka Cluster**
    - Consists of multiple Brokers (servers).
    - Each **Broker** stores data and serves client requests.
    - Data is divided into **Topics** (categories) and further split into **Partitions for scalability.**
3. **Zookeeper**
    - Manages broker metadata.
    - Tracks cluster health and leader election.
4. **Consumers**
    - Applications that **subscribe to topics** and **consume messages** from Kafka.
    - Organized into **Consumer Groups for load balancing** (each consumer reads different partitions

USE CASES OF PYSPARK

Payment transactions streamed in real time for fraud detection.

03 ETL Pipelines & Orchestration

What Are ETL Pipelines?

A data pipeline is like a path that data follows to move from one place to another. In Data Engineering, the most common type is an ETL pipeline:

- **Extract:** Take data from different sources (databases, APIs, logs, apps).
- **Transform:** Clean the data, remove errors, change formats, and prepare it for analysis.
- **Load:** Store the clean, ready data into a target system like a **data warehouse** or **data lake.**

For Example: Imagine an e-commerce company like Amazon:

- **Extract:** Collect sales data from the website, payment gateway, and shipping system.
- **Transform:** Clean duplicates, fix missing values, and standardize currency
- **Load:** Save the clean data into Snowflake, where analysts can run reports.

What Is Orchestration?

Building pipelines is only half the job — you also need to schedule, manage, and monitor them. This is called orchestration.

- Orchestration tools make sure pipelines run in the **right order**, at the **right time**, and **restart automatically** if something fails.
- Without orchestration, you'd have to run everything manually, which is error-prone and inefficient.

Think of orchestration like a conductor of an orchestra: each musician (pipeline task) knows what to play, but the conductor (orchestration tool) ensures they all play in sync.

Apache Airflow

Apache Airflow is the most popular workflow orchestration tool for Data Engineers.

- It is **open-source** and widely adopted across companies.
- Pipelines in Airflow are written as **DAGs (Directed Acyclic Graphs),** where each node is a task (like extract data, transform it, load into warehouse).
- Airflow supports **scheduling** (run a job every day at 2 AM), **monitoring** (see which task failed), and **automation** (retry automatically if something breaks).
- It integrates with almost everything: databases, cloud storage, Spark, Kafka, DBT, and more.

For Example: A retailer can use Airflow to automatically extract daily sales data from an API, run cleaning tasks, and then load it into Snowflake — all without manual work.

Databricks

Databricks is a cloud-based big data and AI platform, built on Apache Spark.

- It allows you to process massive amounts of data — batch or streaming.
- Provides collaborative notebooks (like Google Docs but for code) where teams can write SQL, Python, or R together.
- Ideal for ETL pipelines, advanced analytics, and machine learning.

- Offers integrations with popular storage systems like S3, ADLS, and warehouses like Snowflake.
- Can scale up or down depending on workload, making it flexible for both small and very large tasks.

For Example: Netflix uses Databricks to process billions of daily events (watch, pause, search) in real time and feed their recommendation algorithms.

DBT (Data Build Tool)

DBT is a modern transformation tool that focuses purely on the T in ETL (Transform).

- Unlike Airflow or Databricks, DBT doesn't handle extraction or loading — it only transforms data that's already in a data warehouse.
- It uses SQL for transformations, but organizes queries as modular models (instead of messy long scripts).
- Supports version control (Git), so teams can track changes to transformations.
- Automatically generates data lineage documentation (showing how raw data is turned into final tables).

For Example: A company has raw orders and customers tables in BigQuery. With DBT, they can build a clean sales_summary table by joining, filtering, and aggregating those tables — all version-controlled and documented.

PHASE 5:

Cloud Technologies

Cloud computing is a core part of **modern data engineering** because most organizations now use **cloud platforms** for storage, processing, and orchestration. This phase introduces **AWS, Azure, and GCP** and containerization with **Docker.**

WHY CLOUD FOR DATA ENGINEERS?

- Data pipelines and warehouses run on cloud infrastructure for scalability.
- Cloud services simplify ETL pipelines, data storage, and streaming.
- Essential for handling big data workloads and deploying data solutions.

KEY TOPICS IN THIS PHASE:

01 AWS (Amazon Web Services)

1. **Core Services for Data Engineering:**
    - S3 (Simple Storage Service): Object storage for raw and processed data.
    - EC2: Virtual servers for running processing jobs.
    - RDS: Managed relational databases.
    - Glue: ETL service for big data.
    - Athena: Query S3 data using SQL.
2. **Docker on AWS:** Deploy containerized data processing apps using ECS or EKS.

02 Azure (Microsoft)

1. **Core Services:**
    - Azure Data Lake Storage (ADLS): Stores large-scale data.
    - Azure Synapse Analytics: Cloud data warehouse.
    - Data Factory: Orchestration and ETL tool.
    - Azure Kubernetes Service (AKS): For Dockerized workloads.

03 GCP (Google Cloud Platform)

1. **Core Services:**
    - BigQuery: Serverless data warehouse for analytics.
    - Cloud Storage: Object storage.
    - Dataflow: Stream/batch data processing.
    - Dataproc: Managed Hadoop/Spark clusters.
    - GKE (Google Kubernetes Engine): Container orchestration for Docker apps.

04 Docker

1. **Why Docker?**
    - Packages apps and dependencies into containers.
    - Ensures consistent environments across dev, test, and production.
2. **For Data Engineers:**
    - Containerize Spark jobs, Airflow, or ETL pipelines.
    - Deploy ML/data pipelines on Kubernetes clusters.

ALSO LEARN:

Data Structures & Algorithms for Data Engineers

You don’t need to go as deep as competitive programmers, but you must cover:

- Hashing (dictionaries/sets) → fast lookups, removing duplicates.
- Stacks & Queues → order-based data processing.
- Linked Lists → useful for interview practice, less in real DE.
- Time & Space Complexity → to evaluate how efficient your solution is.

PROJECTS:

Turn Theory into Practice: Data Engineering Projects

1. **Uber Data Analytics Dashboard (End-to-End Pipeline)**
    - **Description:** Build a comprehensive analytics pipeline using Uber trip data. Ingest, transform, and store data in a warehouse or data lake, then design a dashboard for insights like trip patterns, demand spikes, and ride durations.
    - **Why It Matters:** Demonstrates full ETL flow, modeling, and data visualization. Common use case in rideshare platforms and logistics analytics.

2. **Amazon Product Recommendation Analytics**
    - **Description:** Process Amazon review data stored in cloud storage (e.g., AWS S3). Leverage Spark for sentiment analysis, keyword extraction, and categorization. Store results in Redshift and visualize trends (e.g., sentiment over time, recurring issues, feature mentions) using BI tools.
    - **Why It Matters:** Applies big data processing and ETL in a retrievable, analytical pipeline—highly analogous to e-commerce personalization systems

3. **Spotify Music Trends Analysis**
    - **Description:** Analyze music listening trends across users—track top genres, seasonal patterns, or regional preferences. Use large-scale data processing, perhaps leveraging tools like Spark, Kafka, or cloud analytics services.
    - **Why It Matters:** Mirrors how streaming platforms derive insights for content recommendation and user engagement monitoring.

BOSSCODER ACADEMY

WHY BOSSCODER?

01 STRUCTURED INDUSTRY-VETTED CURRICULUM

Our curriculum covers everything you need to get become a skilled software engineer & get placed.

02 1:1 MENTORSHIP SESSIONS

You are assigned a personal mentor currently working in Top product based companies.

03 2200+ ALUMNI PLACEMENT

2200+ Alumni placed at Top Product-based companies.

04 24 LPA AVERAGE PACKAGE

Our Average Placement Package is **24 LPA** and highest is **98 LPA**

Niranjan Bagade
Software Engineer,
British Petroleum

10 Years
Experience

NICE	Hike	British Petroleum
Software Eng. Specialist	83%	Software Engineer

Dheeraj Barik
Software Engineer 2,
Amazon

2 Years
Experience

Infosys	Hike	Amazon
Software Engineer	550%	SDE 2

EXPLORE MORE