UNIT-II

DISTRIBUTED MESSAGING, DATA INGESTION AND STREAM MANAGEMENT

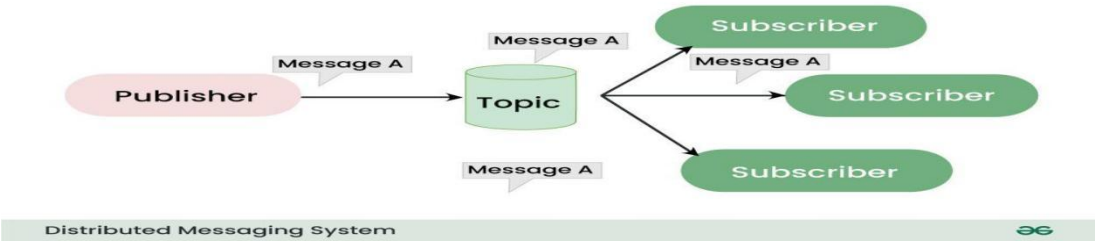
Distributed Messaging Fundamentals – Event Brokers – Data Ingestion Pipelines – Apache Kafka Architecture – Kafka Producers and Consumers – Kafka Streams – Apache Pulsar – Apache Flume – Apache NiFi – Apache ZooKeeper – Schema Registry – Data Serialization (Avro, Protobuf, JSON) – Stream Partitioning – Event Ordering – Data Quality – Data Governance – Data Lineage – Cloud Event Streaming Services.

Distributed Messaging Fundamentals

- *Distributed Messaging Fundamentals are the backbone of modern event-driven and microservice architectures, enabling independent systems to communicate reliably through message passing, queues, and publish-subscribe models.*
- *At their core, they provide decoupling, scalability, and fault tolerance for distributed applications.*

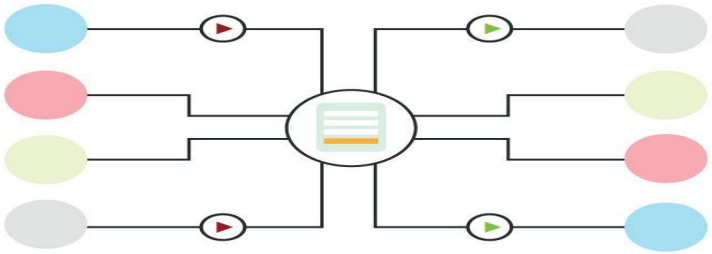
What is a Distributed Messaging System?

- *A Distributed Messaging System is like a smart way of sending messages and information. Instead of relying on just one place to handle all the messages, it spreads the work across many connected places.*



Distributed Messaging System

How Distributed Messaging System Works?

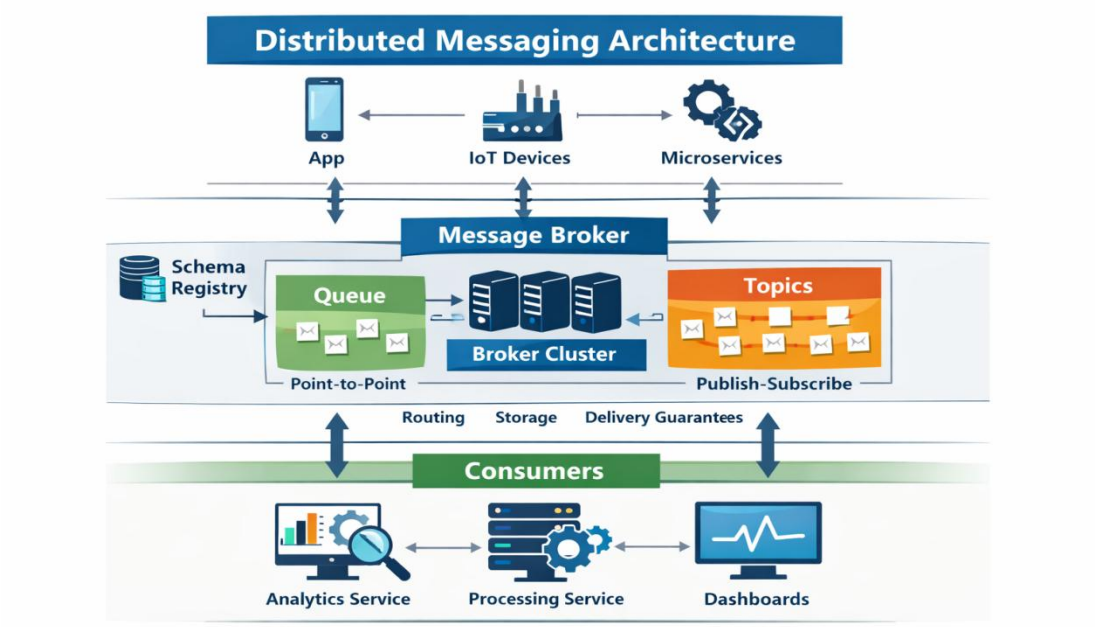


How Distributed Messaging System Works?

- ***Distributed System:*** *Instead of a single node being responsible for disseminating information to all participants, the responsibility is partitioned across multiple nodes. Each node forwards messages to a subset of peers, ensuring scalability and fault tolerance through decentralized propagation*
- ***Messaging System****: Communication occurs via a standardized middleware (e.g., message queues or publish-subscribe topics), which provides a common interface for message exchange. This enables real-time dissemination, consistent state visibility, and reliable delivery across all connected participants.*
- ***Sender****: The producer or publisher that initiates communication by generating and transmitting messages containing event or data payloads.*
- ***Recipients****: The consumers or subscribers that receive and process the messages. In a distributed setup, each consumer is responsible for a defined subset of downstream consumers, enabling parallel dissemination and reducing single-point bottlenecks.*
- ***Messaging System:*** *The middleware (e.g., message broker or pub-sub infrastructure) that provides a unified communication channel. Instead of point-to-point manual delivery, the system ensures reliable, real-time distribution of messages to all subscribed endpoints, maintaining consistency and visibility across the network.*

Core Components

- **Producers:** Applications or services that generate and send messages into the system.
- **Consumers:** Applications or services that subscribe to topics or queues and process incoming messages.
- **Message Brokers:** Middleware (e.g., Kafka, RabbitMQ, Pulsar) that manages message routing, persistence, and delivery guarantees.
- **Topics & Queues:** Logical channels where messages are published and consumed. Topics support pub-sub, queues support point-to-point.



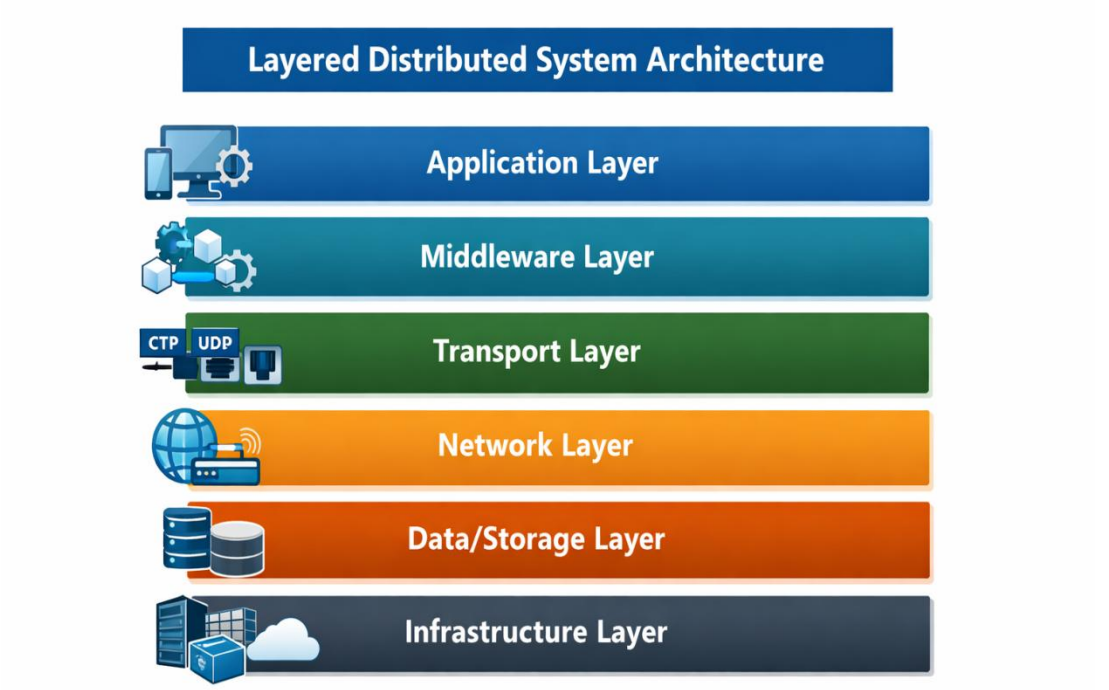
Communication Patterns

- **Point-to-Point**: One producer → one consumer via a queue.
- **Publish-Subscribe**: One producer → multiple consumers via a topic.
- **Fan-out & Routing**: Messages distributed to multiple destinations based on rules or filters.

Delivery Semantics

- **At-Most-Once Delivery** is one of the fundamental message delivery guarantees in distributed messaging systems. It prioritizes performance and simplicity over reliability.
  - each message is delivered zero or one time to the consumer. If a failure occurs during transmission or processing, the message may be lost — but it will never be duplicated.
- **At-Least-Once Delivery** is a reliability model in distributed messaging systems that ensures every message is delivered one or more times to the consumer — guaranteeing that no message is lost, even in the presence of failures.
- **Exactly-Once Delivery** is the strongest guarantee in distributed messaging systems, ensuring that each message is delivered once and only once to the consumer — no loss, no duplication.

<u>Architectural Layers</u>



Core Layers

- Application Layer
  - Interfaces directly with end-users or client applications.
  - Contains business logic, APIs, and services.
  - Example: Web apps, microservices, dashboards.
- Middleware Layer
  - Provides communication and coordination between distributed components.
  - Includes message brokers, RPC frameworks, and service registries.
  - Example: Kafka, RabbitMQ, gRPC, service mesh.
- Transport Layer
  - Ensures reliable data transfer across the network.
  - Implements protocols like TCP/UDP for message delivery.
  - Handles retries, acknowledgments, and flow control.
- Network Layer
  - Responsible for routing and addressing.
  - Uses IP for packet forwarding across distributed nodes.
  - Example: DNS, IP routing, load balancers.
- Data/Storage Layer
  - Provides persistence and consistency for distributed data.
  - Includes distributed databases, logs, and file systems.
  - Example: Cassandra, HDFS, Kafka partitions, Pulsar BookKeeper.
- Infrastructure Layer
  - Physical or virtual resources supporting the system.
  - Includes servers, containers, cloud platforms, and orchestration tools.
  - Example: Kubernetes, Docker, AWS/GCP/Azure.

Layered View

Layer	Role	Examples
Application	User-facing services	Web apps, APIs
Middleware	Communication & coordination	Kafka, RabbitMQ, gRPC
Transport	Reliable delivery	TCP, UDP
Network	Routing & addressing	IP, DNS, Load balancers
Storage	Persistence & consistency	Cassandra, HDFS
Infrastructure	Compute & orchestration	Kubernetes, Docker

Advantages of Using Distributed Messaging Systems

- Fault Tolerance No single point of failure — multiple nodes ensure continuity even if one fails.
- Scalability Handles large workloads efficiently by distributing tasks across nodes.
- Performance Optimization Parallel processing and asynchronous communication enable faster message delivery.
- Resource Efficiency Dynamically allocates resources, minimizing idle capacity and energy consumption.

- Elastic Growth Adapts seamlessly to increasing user demand or data volume without degradation.

Disadvantages of Using Distributed Messaging Systems

- Coordination Complexity Synchronizing multiple nodes and ensuring consistency can be challenging.
- System Complexity Requires careful configuration, monitoring, and maintenance.
- Communication Failures Network latency or partitioning may cause message loss or duplication.
- Security Management Multiple endpoints increase the attack surface; encryption and authentication are essential.
- Operational Cost Infrastructure, monitoring, and redundancy increase deployment costs.

Use Cases of Distributed Messaging Systems

- Real-Time Data Processing – Stream data from IoT devices, sensors, or financial feeds.
- Event-Driven Architecture – Enables reactive systems that respond instantly to events.
- Microservices Communication – Decouples services for independent scaling and fault isolation.
- Asynchronous Task Queues – Offloads non-critical tasks for deferred execution.
- Notification Systems – Powers real-time alerts and updates for users.

Popular Distributed Messaging Systems

System	Key Concepts	Highlights
Apache Kafka	Topics, Brokers, Partitions, ZooKeeper	High throughput, fault-tolerant, durable streaming platform
RabbitMQ	Exchanges, Queues, Bindings, Virtual Hosts	Implements AMQP, reliable and interoperable
Apache ActiveMQ	Destinations, Persistence, Network of Brokers, Message Selectors	JMS-compliant, supports multiple protocols
Amazon SQS	Queues, Visibility Timeout, Long Polling, FIFO Queues	Fully managed, scalable, low-latency AWS service
Apache Pulsar	Topics, Subscriptions, Brokers, Bookies, Schema Registry	Multi-tenancy, durability, and high scalability

EVENT BROKER

What is an event broker?

- An event broker is message-oriented middleware that enables the transmission of events between different components of a system, acting as a mediator between publishers and subscribers.
- It is the cornerstone of event-driven architecture, and all event-driven applications use some form of event broker to send and receive information.
- An event broker can be hardware, software, or software-as-a-service (SaaS).
- A network of event brokers is called an event mesh.



What's the difference between an event broker and message broker

- Event brokers and message brokers both establish loosely coupled relationships between applications using message exchange patterns like publish/subscribe and request/reply, or combine multiple patterns, depending on the specific requirements of the interaction.
- They both support various qualities of services such as non-persistent and persistent delivery through the use of queues, but generally speaking message brokers are better at point-to-point and queue-oriented data exchange, while event brokers are better at real-time publish/subscribe interactions.

What features should an event broker support?

- *Some of the advanced features that an event broker should provide include:*
  - *Stream processing*
  - *Various delivery modes, including "exactly once," "at most once," or "at least once"*
  - *Persistent event ledger for analysis and event sourcing*
  - *Security at various levels of authentication, authorization and topic/channel access control*
  - *Programmable extensibility for custom filtering and analysis*
  - *High availability and disaster recovery*

What are the best event brokers?

- Gartner lists these products as the top 5 best event brokers in their "Event Broker" category for Peer Insights reviews and ratings (2023):
- PubSub+ Platform – by Solace
- Microsoft Azure Event Grid – by Microsoft
- IBM MQ – by IBM
- Confluent Cloud – by Confluent
- TIBCO Messaging – by TIBCO Software

DATA INGESTION PIPELINES

- *Data ingestion pipeline is a crucial component of modern data architecture, enabling businesses to efficiently manage and utilize their data.*
- *It's the process of importing, transferring, loading, and processing data for later use or storage in a database.*
- *This process is integral to data systems, as it's the first step in making raw data accessible and usable for analytics and decision-making.*
- *The data sources can be numerous and diverse, including databases, servers, other data centers, and even online sources.*

Understanding the data ingestion process

- Data ingestion involves taking data from various sources and moving it to a system where it can be stored and analyzed.
- The data sources can be numerous and diverse, including databases, servers, other data centers, and even online sources. The data is then processed and loaded into a destination system, such as a data warehouse or a data lake, where it can be accessed and analyzed by data scientists and other users

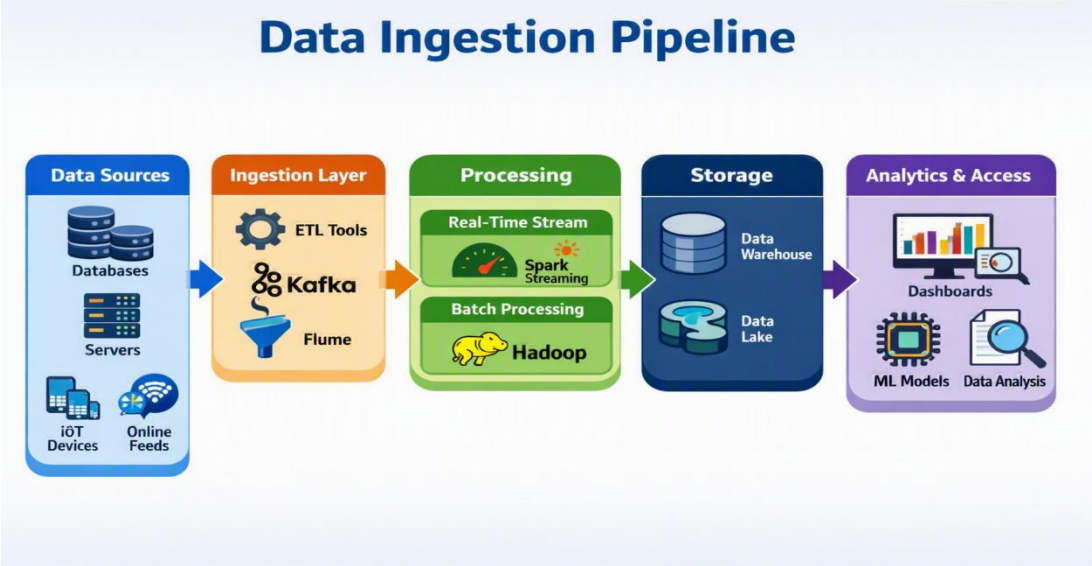
What are the different types of data ingestion?

- There are primarily two types of data ingestion methods:
  - real-time
  - batch.
- **Real-time data ingestion** involves importing data as soon as it is produced, providing users with up-to-date, real-time insights. This method is crucial for businesses that need to make immediate decisions based on the latest data.
- **Batch data ingestion**, on the other hand, involves collecting and importing data at regular intervals, whether it's every few hours, once a day, or once a week. This method is suitable for businesses that do not require real-time data and can make decisions based on periodic data updates.

Importance

- Enables centralized access for data scientists and analysts.
- Provides foundation for analytics, machine learning, and business intelligence.

- Supports decision-making with timely insights.



Process Flow

- Data Sources → Databases, servers, IoT, online feeds.
- Ingestion Layer → ETL tools, Kafka, Flume.
- Processing → Real-time (Spark Streaming) or batch (Hadoop).
- Storage → Data warehouse, data lake.
- Access → Analysts, dashboards, ML models.

benefits of data ingestion

- Data ingestion offers several benefits:
- Efficiency: Automated data ingestion pipelines save time and resources by streamlining the process of importing, processing, and storing data.
- Reliability: These pipelines ensure that the data is consistently and accurately imported, making it reliable for analysis.
- Timely Insights: Real-time data ingestion pipelines provide up-to-date insights, enabling businesses to make timely decisions.
- Scalability: As businesses grow, so does the amount of data they generate. Data ingestion pipelines can scale to handle increasing data volumes.

Building an effective data ingestion pipeline

- Building an effective data ingestion pipeline involves several key steps:
- Identify the Data Sources: The first step in building a data ingestion pipeline is to identify the data sources. These could be databases, online sources, servers, or other data centers.

- Determine the Destination System: The next step is to determine where the data will be stored and analyzed. This could be a data warehouse, a data lake, or another type of data storage system.
- Choose the Data Ingestion Method: The data ingestion method could be either real-time or batch, depending on the needs of the business.
- Design the Data Ingestion Process: This involves designing the process of importing, processing, and loading the data into the destination system. This process should be automated to ensure consistency and accuracy.
- Monitor and Optimize the Data Ingestion Pipeline: Once the data ingestion pipeline is in place, it's important to monitor its performance and make any necessary adjustments to ensure it's running efficiently.

Data ingestion and ELT

- Extract, Load, Transform (ELT) is a modern approach to data integration that focuses on faster data availability, flexibility, and scalability.
- In an ELT pipeline, data is extracted from sources, loaded into storage, and then transformed as needed.
- This approach decouples the extraction and transformation processes, allowing for quicker data ingestion and giving analysts and data scientists faster access to data.

Data ingestion approaches

- There are two main approaches to data ingestion: manual coding and data integration platforms.
- Manual coding involves engineers and developers writing each line of code required to build a data pipeline. This approach is time-consuming and labor-intensive.
- Data integration platforms, on the other hand, provide pre-built connectors and transformations to streamline the data pipeline process. These platforms are fully managed, reducing the need for constant updates and maintenance.