UNIT-5

Cloud Storage

1. Introduction to Storage Systems

Definition

A **storage system** refers to the combination of **hardware, software, and processes** used to **store, manage, protect, and retrieve digital data** efficiently.
It plays a critical role in computing environments—from personal devices to large-scale enterprise and cloud infrastructures.

In **cloud computing**, storage is delivered as a **service over the internet**, allowing users and organizations to **store data remotely** and access it **on demand** without managing physical hardware.
This concept is often referred to as **Storage-as-a-Service (SaaS)**.

Objectives of Storage Systems

- To **store large volumes of data** securely and reliably.
- To **provide fast and consistent access** to data.
- To **ensure data durability** (protection against loss or corruption).
- To **scale easily** as data volumes grow.
- To **support backup and disaster recovery**.

Components of a Storage System

1. **Storage Devices** – Physical media such as hard disk drives (HDDs), solid-state drives (SSDs), or magnetic tapes.
2. **Storage Controllers** – Manage the flow of data between storage devices and host systems.
3. **Storage Software** – Provides features like data replication, compression, deduplication, and encryption.
4. **Interconnects** – Communication links like Fibre Channel, Ethernet, or SAS used to transfer data between storage and compute nodes.

Types of Storage Systems

1. Direct Attached Storage (DAS)

- **Definition:**
  Storage that is **physically connected** to a single computer or server via interfaces like SATA, SCSI, or NVMe.
- **Characteristics:**
  - Provides **high-speed local access** to data.
  - **Simple and low-cost** to implement.

    - **Limited scalability**—data cannot be easily shared between multiple systems.
- **Examples:**
    - Internal hard drives in desktops or laptops.
    - External USB drives or SSDs connected to servers.
- **Use Cases:**
    - Suitable for **single-user** or **standalone applications**.
    - Common in **small businesses** or **testing environments**.

2. Network Attached Storage (NAS)

- **Definition:**
  A **dedicated file storage system** connected to a network that allows multiple clients or users to access shared files.
- **Characteristics:**
    - Operates at the **file level** (users access files and directories).
    - Uses **standard network protocols** such as:
        - **NFS (Network File System)** for UNIX/Linux.
        - **SMB/CIFS (Server Message Block / Common Internet File System)** for Windows.
    - Centralized storage accessible by multiple systems.
    - Easier to **expand** than DAS.
- **Examples:**
    - Synology NAS, QNAP NAS, enterprise NAS servers.
- **Use Cases:**
    - File sharing, collaboration, and **shared data repositories** in offices and organizations.
    - **Backup and archival storage**.

3. Storage Area Network (SAN)

- **Definition:**
  A **high-speed network** that provides **block-level access** to storage.
  Unlike NAS, SAN appears to the operating system as a **locally attached disk** even though it is remote.
- **Characteristics:**
    - Provides **high performance** and **low latency**.
    - Supports **large-scale storage pooling** and **data redundancy**.
    - Uses **Fibre Channel (FC)** or **iSCSI** protocols.
- **Examples:**
    - EMC SAN, NetApp SAN solutions, Dell PowerVault.
- **Use Cases:**
    - Enterprise data centers.
    - Applications requiring **high-speed access**, such as **databases, virtualization, and transaction processing**.

4. Cloud Storage

- **Definition:**
  Data storage provided as a **cloud service** over the internet, managed by a **cloud service provider (CSP)**.

- **Characteristics:**
  - Fully **managed, scalable, and elastic** — users pay only for what they use.
  - **Accessible from anywhere** via the internet.
  - Offers **durability and redundancy** through replication across multiple data centers.
  - Integrated with **security features**, such as encryption and access controls.
- **Examples:**
  - **Amazon Web Services (AWS) S3**
  - **Google Cloud Storage**
  - **Microsoft Azure Blob Storage**
  - **Dropbox, OneDrive**
- **Use Cases:**
  - Data backups and disaster recovery.
  - Web application data storage.
  - Big data analytics and machine learning datasets.

Comparison of Storage Systems

Feature	DAS	NAS	SAN	Cloud Storage
Access Type	Local	File-level	Block-level	Object/File/Block
Connectivity	Direct (e.g., SATA, USB)	Network (Ethernet)	Dedicated network (Fibre Channel/iSCSI)	Internet
Scalability	Low	Moderate	High	Very High
Performance	High (local)	Moderate	Very High	Depends on internet/network
Management	Local	Centralized	Centralized	Provider-managed
Cost	Low	Medium	High	Pay-as-you-go

2. Data in the Cloud: Relational Databases

Introduction

In the world of cloud computing, data storage and management are key components. Among various data storage models, **relational databases** remain one of the most widely used due to their **structured approach, reliability, and strong consistency**.

A **cloud-based relational database** combines the traditional relational model with the scalability and flexibility of cloud infrastructure.

It allows users to **store, access, and manage structured data** over the internet without worrying about hardware setup, maintenance, or database administration tasks.

Relational Database Concepts

1. Definition

A **relational database (RDB)** is a type of database that stores data in **tables (relations)** consisting of **rows and columns**.
Each row (record) represents a single entry, and each column (field) represents a data attribute.

- **Example:**
  A *Student* table may include columns like *Student_ID, Name, Course, Marks*, etc.

Student_ID	Name	Course	Marks
101	Rahul	Cloud Computing	85
102	Neha	AI	90

Here:

- Each **row** is a record (one student).
- Each **column** represents a data field (like Name or Marks).

2. Structured Query Language (SQL)

- SQL is the **standard language** used to **query, insert, update, and manage data** in relational databases.
- Examples of SQL commands:
  - `SELECT * FROM Students;`
  - `INSERT INTO Students VALUES (103, 'Amit', 'Data Science', 88);`
  - `UPDATE Students SET Marks = 92 WHERE Student_ID = 102;`

SQL provides a powerful, declarative way to manipulate and retrieve data efficiently.

3. ACID Properties

Relational databases maintain **ACID properties** to ensure reliability and integrity of transactions:

Property	Description
A – Atomicity	A transaction is *all or nothing*. If one operation fails, the entire transaction rolls back.

Property	Description
C – Consistency	Ensures data follows defined rules and constraints, maintaining accuracy.
I – Isolation	Transactions are executed independently to prevent interference.
D – Durability	Once a transaction is committed, it remains saved even after system failure.

These properties make relational databases ideal for **financial, business, and mission-critical applications**.

Cloud-Based Relational Databases

Cloud-based relational databases are **managed database services** hosted on cloud platforms. They provide **database functionality without manual setup**, hardware provisioning, or administrative overhead.

Instead of installing a database server manually, users simply create a database instance via a **web console or API**, and the cloud provider handles:

- Software installation and patching
- Backups and replication
- Scaling and monitoring
- High availability

Popular Cloud Relational Database Services

1. Amazon RDS (Relational Database Service)

- A fully managed service by **Amazon Web Services (AWS)**.
- Supports multiple database engines:
  - MySQL
  - PostgreSQL
  - MariaDB
  - Oracle
  - Microsoft SQL Server
  - Amazon Aurora (AWS's high-performance engine)
- **Features:**
  - Automated backups, monitoring, and patching.
  - Multi-AZ (Availability Zone) replication for **high availability**.
  - Easy scaling of compute and storage.

2. Google Cloud SQL

- A **managed relational database service** by **Google Cloud Platform (GCP)**.
- Supports:

    - MySQL
    - PostgreSQL
    - SQL Server
- **Features:**
    - Automatic replication and failover.
    - Point-in-time recovery.
    - Integration with **BigQuery** and **Google App Engine** for analytics and application development.

3. Microsoft Azure SQL Database

- A **cloud-based version of Microsoft SQL Server** managed by **Microsoft Azure**.
- **Features:**
    - Built-in **AI-powered performance tuning**.
    - **Geo-replication** for disaster recovery.
    - **Serverless mode** that automatically scales compute resources based on workload.

4. Other Examples

- **Oracle Autonomous Database (Oracle Cloud)** – Self-tuning, self-patching, and self-securing RDBMS.
- **IBM Db2 on Cloud** – Fully managed enterprise-grade RDBMS solution.

Advantages of Cloud-Based Relational Databases

Advantage	Description
Managed Service	Cloud provider handles database setup, patching, monitoring, and maintenance.
Automatic Backups	Regular snapshots and recovery options ensure data protection.
High Availability	Redundant infrastructure across multiple zones prevents downtime.
Scalability	Storage and compute can scale up or down based on demand.
Cost Efficiency	Pay only for resources used (pay-as-you-go model).
Security and Compliance	Built-in encryption, access control, and compliance certifications (GDPR, HIPAA, etc.).
Integration	Easily integrates with other cloud services (analytics, machine learning, etc.).

Limitations of Cloud Relational Databases

Limitation	Explanation
Not ideal for unstructured data	Relational databases work best with structured, tabular data — not documents, images, or sensor data.
Scalability challenges	Traditional RDBMS have limitations in horizontal scaling (cannot easily distribute across many servers).
Latency	Accessing cloud data over the internet may introduce small delays compared to local databases.
Vendor lock-in	Migration between cloud providers can be complex.
Cost management	Improper resource allocation (e.g., leaving large instances running) can increase costs.

Use Cases of Cloud Relational Databases

- **E-commerce systems** (orders, customers, transactions).
- **Banking and finance** (accounts, payments, ledgers).
- **ERP and CRM systems** (customer and business data).
- **Web and mobile applications** needing structured, consistent data management.

3. Cloud Storage Concepts

Introduction

Cloud Storage is a model of data storage where digital data is stored in logical pools and hosted on **remote servers** accessed via the **internet**.
The data is managed, maintained, and backed up by a **Cloud Service Provider (CSP)**, such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud.

Instead of managing physical storage hardware, organizations can store unlimited data in the cloud, **pay only for what they use**, and access it from anywhere at any time.

Key Concepts in Cloud Storage

1. Elasticity and Scalability

- **Elasticity** refers to the ability of a cloud storage system to **automatically adjust storage capacity** based on current demand.
- **Scalability** means users can **increase or decrease storage resources** easily without service interruption.

Example:

- During high traffic periods (e.g., festive e-commerce sales), storage can automatically expand to handle more user data.
- AWS S3 and Google Cloud Storage automatically scale without manual intervention.

Benefits:

- Eliminates over-provisioning or under-provisioning.
- Optimizes cost and performance.

2. Durability

- **Durability** ensures that stored data is **protected against loss or corruption** over time.
- Cloud providers achieve this by **replicating data across multiple storage devices and regions**.
- Even if a server or disk fails, another copy remains accessible.

Example:

- **Amazon S3** promises **99.999999999% (11 nines) durability**, meaning data loss is nearly impossible.
- Data blocks are continuously verified for integrity.

Key Mechanisms:

- Data replication
- Error correction codes
- Regular integrity checks

3. Availability

- **Availability** ensures that users can **access their data at all times**, even in the event of hardware or network failures.
- Achieved through **redundancy**, **load balancing**, and **fault tolerance**.

Example:

- AWS and Azure provide **multi-zone replication**, so if one data center goes down, another instantly serves the data.

Metrics:

- Measured as a percentage uptime (e.g., 99.99% availability).
- Includes automatic failover and data recovery systems.

4. Security

- Cloud storage providers implement multiple **security layers** to protect data from unauthorized access and breaches.

Security Mechanisms:

1. **Encryption:**
    - **At rest:** Data stored on disks is encrypted using strong algorithms (e.g., AES-256).
    - **In transit:** Data moving over networks uses secure protocols (e.g., TLS/SSL).
2. **Access Control:**
    - Identity and Access Management (IAM) policies control who can access data.
3. **Authentication and Authorization:**
    - Multi-factor authentication (MFA), tokens, and role-based permissions ensure only verified users can interact with data.
4. **Auditing and Monitoring:**
    - Continuous tracking of data access logs for suspicious activities.

Example:
AWS Key Management Service (KMS) manages encryption keys securely.

5. Multi-Tenancy

- **Multi-tenancy** means multiple users or organizations **share the same physical infrastructure** but have **logically isolated** environments.
- Each user's data and configurations remain private and secure.

Example:
In AWS S3, many customers' data may be stored on the same physical servers, but each account's data is isolated through virtual partitions and access policies.

Benefits:

- Efficient resource utilization.
- Lower costs through shared infrastructure.

6. Data Consistency Models

When multiple users access and modify cloud data, **consistency models** define how updates are propagated and reflected.

Consistency Model	Description	Example Use
Strong Consistency	After a write, any subsequent read returns the latest data immediately.	Banking transactions, ledgers

Consistency Model	Description	Example Use
Eventual Consistency	Updates are propagated asynchronously; data may temporarily appear outdated. Eventually, all nodes become consistent.	Social media posts, log systems

Example:
AWS S3 provides **strong read-after-write consistency** for new objects but may use **eventual consistency** for overwrite or delete operations.

4. Storage Types in the Cloud

Cloud storage is offered in three primary models based on data structure and use case: **Object Storage, Block Storage, and File Storage.**

1. Object Storage

- Stores data as **objects**, each consisting of:
  - The **data itself** (e.g., image, video, document)
  - **Metadata** (information about the data)
  - A **unique identifier (Object ID)**
- Data is stored in a **flat structure** (no folders), making it highly scalable and suitable for **unstructured data**.

Examples:

- **Amazon S3 (Simple Storage Service)**
- **Google Cloud Storage**
- **Azure Blob Storage**

Features:

- Ideal for backups, multimedia files, logs, and archives.
- Supports **versioning**, **metadata tagging**, and **global accessibility**.
- Accessed via HTTP/HTTPS APIs (RESTful).

Use Cases:

- Media content delivery
- Data archiving
- Big data analytics storage

2. Block Storage

- Data is divided into **fixed-size blocks**, each with a unique address.

- Works like a **hard drive** — the operating system manages files on top of these blocks.

Examples:

- **Amazon EBS (Elastic Block Store)**
- **Azure Managed Disks**
- **Google Persistent Disk**

Features:

- Offers **high performance** and **low latency**, suitable for databases or virtual machines (VMs).
- Provides **consistent read/write speeds**.
- Each block can be accessed and modified independently.

Use Cases:

- Relational databases (e.g., MySQL, PostgreSQL)
- Virtual machine disks
- Transaction-heavy workloads

3. File Storage

- Stores data in a **hierarchical structure** of directories and files (similar to traditional file systems).
- Supports standard file protocols like **NFS (Network File System)** and **SMB (Server Message Block)**.

Examples:

- **Amazon EFS (Elastic File System)**
- **Azure Files**
- **Google Filestore**

Features:

- Easy sharing across multiple instances.
- Good for legacy applications that require file-level access.
- Automatically scales as data grows.

Use Cases:

- Shared file repositories
- Web content management
- Enterprise applications

Comparison of Cloud Storage Types

Feature	Object Storage	Block Storage	File Storage
Structure	Flat (objects with metadata)	Fixed-size blocks	Hierarchical (files/folders)
Access Method	REST API	Low-level block access	File protocols (NFS/SMB)
Best For	Unstructured data, backups	Databases, VMs	Shared file systems
Scalability	Very High	Moderate	Moderate
Performance	High (throughput)	Very High (IOPS)	Moderate
Examples	AWS S3, GCS, Azure Blob	AWS EBS, Azure Disks	AWS EFS, Azure Files

5. Cloud File Systems

Introduction

A **cloud file system** is a distributed storage system designed to handle **massive amounts of data** across multiple servers or data centers in the cloud.
It allows users and applications to **store, access, and process data** efficiently, even when the data is spread over thousands of machines.

Two of the most influential distributed file systems used in cloud and big data environments are:

- **Google File System (GFS)**
- **Hadoop Distributed File System (HDFS)**

Both systems are designed to:

- Store **large datasets** reliably.
- Support **parallel processing** of data.
- Offer **fault tolerance** and **scalability** in distributed environments.

1. Google File System (GFS)

Overview

The **Google File System (GFS)** was developed by **Google** to meet its internal needs for **large-scale data processing** — such as web indexing, crawling, and data analytics.

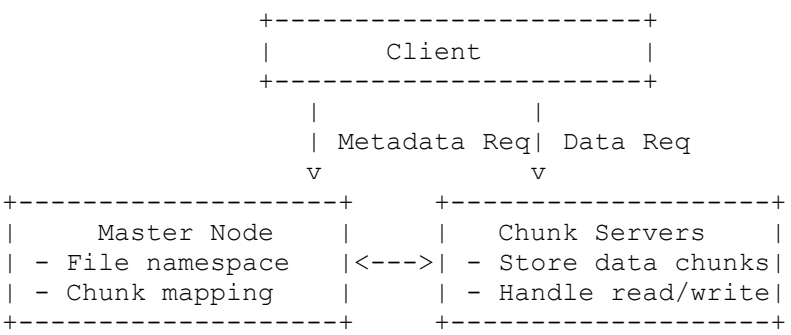
Traditional file systems could not handle Google's massive and ever-growing datasets, so GFS was designed for:

- High throughput over latency.
- Fault tolerance.
- Handling large sequential reads/writes rather than small random I/O operations.

Key Features of GFS

1. **Optimized for Large Files**
   - GFS is designed for files ranging from hundreds of MBs to several GBs.
   - Operations are optimized for **large sequential reads and writes** rather than small updates.
2. **Chunk-based Storage**
   - Files are divided into **fixed-size chunks**, typically **64 MB** each.
   - Each chunk has a **unique 64-bit ID**.
   - Chunks are stored on multiple **chunk servers**.
3. **Master Node Architecture**
   - **Master Node**:
     - Manages all **metadata** (file namespace, chunk mapping, access control, etc.).
     - Keeps track of which chunk servers store each chunk.
   - **Chunk Servers**:
     - Store the actual **data chunks**.
     - Serve read/write requests from clients.
4. **Fault Tolerance via Replication**
   - Each chunk is typically **replicated three times** across different chunk servers.
   - If one server fails, data can be read from another copy.
   - The master node continuously monitors chunk servers and re-replicates chunks if needed.
5. **High Throughput**
   - GFS focuses on **throughput** (total data processed per second) rather than latency.
   - Clients communicate directly with chunk servers for data transfer, minimizing bottlenecks.
6. **Scalability**
   - GFS can scale across **thousands of commodity servers** and **petabytes of data**.

GFS Architecture (Text Diagram)



Use Cases

- Foundation for **Google's MapReduce** framework.
- Used for **Big Data processing**, **web indexing**, and **log analysis**.
- Influenced many other distributed file systems (like HDFS).

2. Hadoop Distributed File System (HDFS)

Overview

The **Hadoop Distributed File System (HDFS)** is an **open-source** distributed file system inspired by GFS.
It was developed as part of the **Apache Hadoop project** to store and process large datasets in a distributed computing environment.

HDFS provides **high throughput**, **scalability**, and **fault tolerance**, making it ideal for **Big Data analytics**.

Architecture of HDFS

HDFS uses a **master-slave architecture** consisting of:

1. **NameNode (Master)**
    - Manages **metadata** and the **directory structure** of the file system.
    - Keeps track of:
        - File names and locations.
        - Which DataNodes hold which blocks.
    - Does **not** store the actual file data.
2. **DataNodes (Slaves)**
    - Store the **actual data blocks** of files.
    - Periodically send **heartbeats** and **block reports** to the NameNode to confirm availability.
3. **Secondary NameNode**
    - A **checkpoint node** that periodically merges the NameNode's metadata logs to prevent loss in case of failure.
    - It is **not** a backup NameNode but helps in recovery.

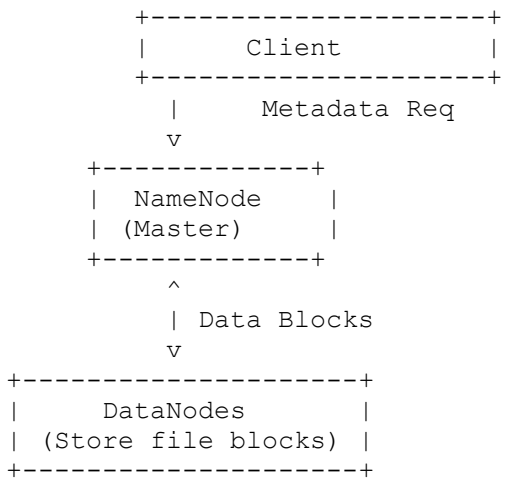
Key Features of HDFS

1. **Block-based Storage**
    - Files are split into **fixed-size blocks** (default: **128 MB or 256 MB**).
    - Each block is stored on multiple DataNodes.
2. **Replication for Fault Tolerance**
    - Each block is **replicated (default: 3 copies)** across different DataNodes.
    - If one node fails, data is read from another replica.
    - Replication ensures **high availability and durability**.
3. **Streaming Data Access**

- Designed for **high throughput** of large data files.
- Optimized for **batch processing** rather than random access.

4. **Scalability**
   - Can easily scale by adding new DataNodes.
   - Suitable for **petabyte-scale** data storage.
5. **Data Locality**
   - Computation is moved **closer to the data** (via MapReduce) to reduce network congestion and improve performance.
6. **Integration**
   - Core storage component of the **Hadoop ecosystem**, used with tools like **Hive**, **Pig**, **Spark**, and **HBase**.

HDFS Architecture (Text Diagram)



Use Cases of HDFS

- **Big Data Analytics** (with Hadoop MapReduce, Spark).
- **Data Warehousing** (Hive, Pig).
- **Machine Learning pipelines**.
- **Log storage and processing** at scale.

Comparison between GFS and HDFS

Feature	GFS	HDFS
Developer	Google	Apache Foundation
Source Type	Proprietary	Open-source
Master Node	Master	NameNode
Data Storage Units	Chunks (64 MB)	Blocks (128/256 MB)

Feature	GFS	HDFS
Replication	Typically 3 copies	Typically 3 copies
Consistency Model	Relaxed consistency	Relaxed consistency
Target Use	Google's internal processing (MapReduce)	General Big Data processing (Hadoop ecosystem)
Fault Tolerance	Through replication	Through replication
Data Access Pattern	Large sequential reads/writes	Streaming access for big data
Scalability	Very high	Very high
Open for Public Use	No	Yes

6. Cloud Databases

Overview

A **cloud database** is a **database service built, hosted, and managed on cloud computing platforms**.
It provides the same functionality as traditional on-premise databases but with the added benefits of **scalability, high availability, and managed infrastructure**.

Cloud databases can be:

- **Relational (SQL-based):** e.g., MySQL, PostgreSQL (managed through AWS RDS, Azure SQL, etc.)
- **Non-relational (NoSQL-based):** e.g., MongoDB, Cassandra, DynamoDB, HBase

These databases are designed to **scale horizontally**, handle **massive workloads**, and support **real-time applications** in a distributed cloud environment.

Key Characteristics of Cloud Databases

1. **Scalability:**
   Automatically adjust storage and compute resources as data or traffic grows.
2. **Availability and Fault Tolerance:**
   Data is replicated across multiple zones or regions to ensure uptime.
3. **Elasticity:**
   Pay-as-you-go model allows automatic resource scaling up or down based on usage.

4. **Global Accessibility:**
   Accessible from anywhere through APIs or web interfaces.
5. **Managed Services:**
   Cloud providers handle backups, updates, patching, and hardware management.
6. **Security:**
   Includes encryption, access control, and compliance with standards (e.g., GDPR, HIPAA).

Types of Cloud Databases

Cloud databases can be broadly divided into:

1. **Relational Databases (SQL):** Structured tables, fixed schema, and ACID transactions.
   Examples: Amazon RDS, Google Cloud SQL, Azure SQL Database.
2. **Non-Relational Databases (NoSQL):** Flexible schema, horizontal scaling, and high performance.
   Examples: HBase, MongoDB, Cassandra, DynamoDB.

Below are detailed explanations of four popular **NoSQL cloud databases**:

A. HBase

Type: Column-oriented NoSQL Database

Built On: Hadoop Distributed File System (HDFS)

Modeled After: Google Bigtable

Overview

Apache HBase is an open-source, distributed, **column-family-based database** that provides **real-time read/write access to large datasets**.
It is part of the **Hadoop ecosystem** and leverages **HDFS** for storage and **MapReduce** for data processing.

Architecture

- **HMaster:** Manages region servers and metadata.
- **Region Servers:** Store and serve subsets of the table's data (regions).
- **Zookeeper:** Coordinates distributed nodes and maintains synchronization.

Features

- **Scalable and Distributed:** Can handle billions of rows and millions of columns.
- **Real-time Access:** Supports low-latency reads/writes to Big Data.
- **Fault Tolerant:** Data replicated across multiple servers using HDFS.
- **Schema-less Columns:** Each row can have variable columns.
- **Integration:** Works seamlessly with Hadoop tools like Hive, Pig, and Spark.

Use Cases

- Real-time analytics on large datasets.
- Log and sensor data storage.
- Time-series and IoT applications.
- Back-end for Big Data platforms.

B. MongoDB

Type: Document-oriented NoSQL Database

Data Model: JSON-like documents stored in **BSON (Binary JSON)** format.

Overview

MongoDB is a **flexible, schema-less database** designed for modern applications that need to store **semi-structured or unstructured data**.
Instead of tables and rows, it uses **collections** and **documents**.

Structure Example

```
{
  "name": "Amit",
  "age": 25,
  "department": "IT",
  "skills": ["Python", "Cloud", "AI"]
}
```

Each document can have different fields — making MongoDB highly flexible.

Features

- **Schema-less Design:** No fixed schema; allows flexible document structures.
- **Indexing & Aggregation:** Efficient querying and analytics via aggregation pipelines.
- **Replication:** Provides high availability through replica sets.
- **Sharding:** Horizontal partitioning for distributed scalability.
- **Integration:** Works well with web, mobile, and IoT apps.

Use Cases

- Web and mobile applications (e.g., user profiles, CMS).
- Real-time analytics and social media apps.
- IoT platforms storing device data.
- Content management systems.

C. Cassandra

Type: Wide-column NoSQL Database

Developed By: Facebook (now Apache Cassandra)

Overview

Apache Cassandra is a **highly scalable and distributed database system** built for **handling large amounts of data** across many commodity servers.
It provides **high availability**, **no single point of failure**, and **tunable consistency**.

Architecture

- **Peer-to-Peer Model:** Every node is equal (no master-slave).
- **Partitioner:** Distributes data evenly using consistent hashing.
- **Replication:** Copies data across multiple nodes for redundancy.
- **Gossip Protocol:** Nodes share status information for coordination.

Features

- **Decentralized Architecture:** No single point of failure.
- **Tunable Consistency:** Users can choose between **strong** or **eventual** consistency.
- **High Write Throughput:** Designed for write-heavy applications.
- **Linear Scalability:** Performance increases proportionally with added nodes.
- **Fault Tolerance:** Data automatically replicated across data centers.

Use Cases

- Time-series data (e.g., IoT sensor readings).
- E-commerce and recommendation engines.
- Messaging and chat applications.
- Event logging and analytics.

D. DynamoDB

Type: Key-Value and Document-oriented NoSQL Database

Developed By: Amazon Web Services (AWS)

Overview

Amazon DynamoDB is a **fully managed, serverless NoSQL database service** that provides **high performance**, **auto-scaling**, and **low latency**.
It is designed for applications requiring **fast and predictable performance** at any scale.

Architecture

- **Tables:** Contain items (similar to rows) with key-value pairs.
- **Primary Keys:** Partition key and optional sort key determine data placement.
- **DAX (DynamoDB Accelerator):** In-memory caching layer for faster reads.
- **Global Tables:** Enable multi-region replication for high availability.

Features

- **Fully Managed:** No server setup, patching, or maintenance needed.

- **Automatic Scaling:** Adjusts read/write throughput and storage automatically.
- **Low Latency:** Typically single-digit millisecond response times.
- **Flexible Data Models:** Supports both key-value and document storage.
- **Integration with AWS Ecosystem:** Works seamlessly with Lambda, S3, and Redshift.

Use Cases

- Real-time analytics and dashboards.
- Gaming leaderboards and player state management.
- IoT data ingestion and tracking.
- E-commerce order tracking.
- Serverless applications.

Comparison of Cloud Databases

Feature	HBase	MongoDB	Cassandra	DynamoDB
Type	Column-oriented	Document-oriented	Wide-column	Key-value / Document
Model	Based on Bigtable	BSON Documents	Peer-to-peer	Managed AWS Service
Storage Layer	HDFS	Native Filesystem	Distributed Nodes	AWS Cloud
Scalability	High	High (sharding)	Very High	Automatic
Consistency	Strong	Eventual (by default)	Tunable	Tunable
Availability	High	High	Very High	Very High
Best For	Big Data Analytics	Web & Mobile Apps	Time-Series, Messaging	Real-Time Apps, IoT
Management	Self-managed	Self-managed or Atlas	Self-managed	Fully Managed