Cloud Storage and Databases Fundamentals

On this page

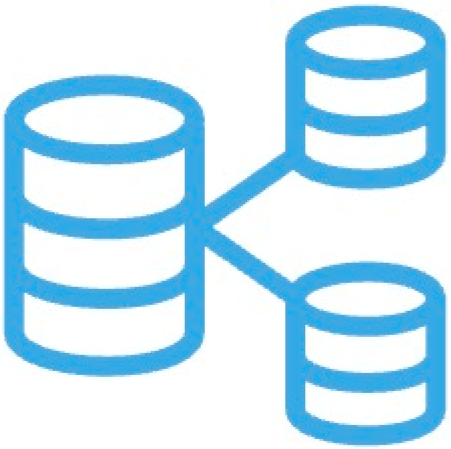
What is Object Storage?

What is Hybrid Storage?

What is the need for Data Analytics?

Compare Data Warehouse vs Data Lake

Compare Availability vs Durability vs Consistency



Availability

- **Definition**: Will your application or data be accessible **when you need it**?
- **Measured As**: Uptime percentage over time (e.g., per month or year)
- **Typical Targets**:
  - **99.95%** → ~22 minutes of downtime/month
  - **99.99% (4 9's)** → ~4.5 minutes/month
  - **99.999% (5 9's)** → ~26 seconds/month
- **How to Improve**:
  - Have **multiple** copies/instances
  - Use **multi-zone or multi-region deployments**
  - Add **load balancers** and **health checks**
  - Implement **replication** and **auto-failover**

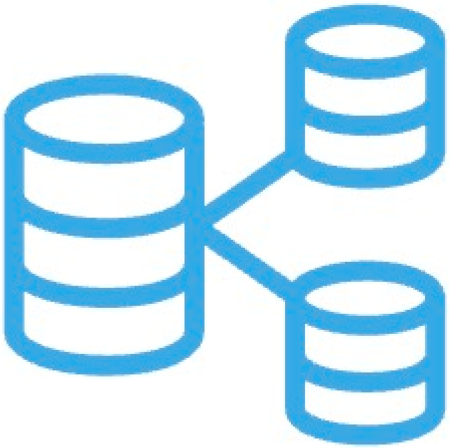
Durability

- **Definition**: Will your data **still exist** years from now, even through hardware failures or disasters?
- **Measured As**: Probability of data loss over time
- **Typical Targets**:
  - **99.999999999% (11 9's)** durability
  - Means: Store **1 million files for 10 million years**, lose **only 1 file**
- **Why It Matters**:

    - Once data is lost, it cannot be recovered
    - Critical for financial data, medical records, backups, archives,
- **How to Improve**:
    - Store **multiple copies** of data
    - Distribute copies across **zones and regions**

Availability vs Durability

Concept	Focus	Metric Example	Goal
Availability	Data is accessible now	99.99% (4 9's) uptime	Avoid downtime
Durability	Data is never lost	99.999999999% (11 9's)	Prevent permanent data loss



What is Consistency?

- **Scenario**: You update your data. Should that update be visible in **all replicas immediately**, or is it okay if some replicas take a few seconds to catch up?
- **Goal**: Choose the right **consistency model** based on the balance between **speed** and **data accuracy**.

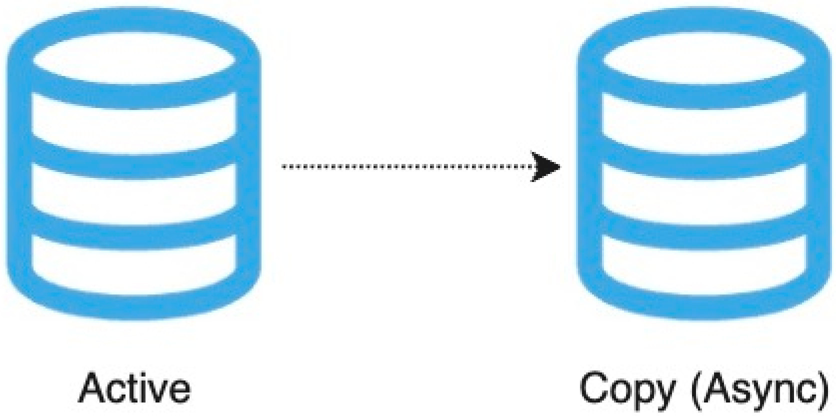
Examples of Consistency Models

1. Strong Consistency

- **Definition**: All replicas return the **same, most recent value** after a write
- **How It Works**: Synchronous replication — write must complete in all replicas before confirming success
- **Use Case**: Banking transactions, inventory systems
- **Trade-offs**:
  - **High integrity**, but **slower performance**

2. Eventual Consistency

- **Definition**: All replicas will **eventually** reflect the latest value — but may show different values for a short period
- **How It Works**: Asynchronous replication — write completes in one node, and propagates later
- **Use Case**: Social media posts, product reviews, user feeds
- **Trade-offs**:
    - **High performance and scalability**, but **temporary inconsistency**
    - Suitable when **speed** is more important than **immediate accuracy**



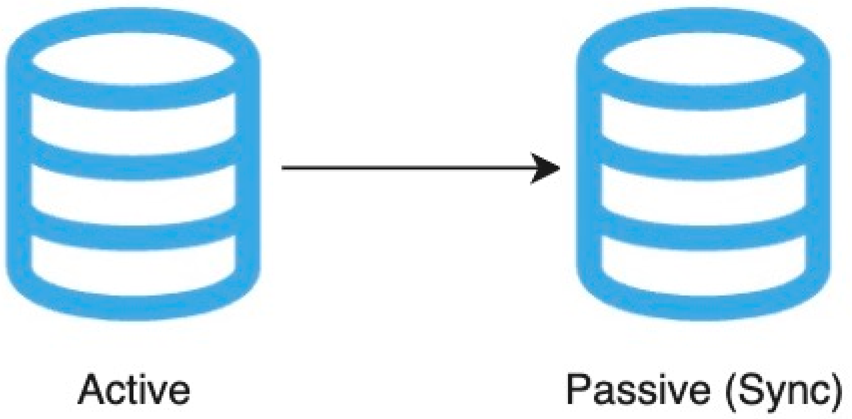
Choose the right database based on your Consistency requirement

Consistency Model	Example Databases
Strong Consistency	Cloud Spanner, Azure SQL DB, Amazon Aurora, Amazon RDS
Eventual Consistency	Amazon DynamoDB (default), Azure Cosmos DB (supports eventual)

Compare RTO vs RPO

RTO vs RPO – What’s the Difference?

- **Scenario**: If a system crashes or data is lost, your business wants to know: *How soon can we recover?* and *How much data can we afford to lose?*
- **Goal**: Understand and define **Recovery Time Objective (RTO)** and **Recovery Point Objective (RPO)** for your applications



RTO (Recovery Time Objective)

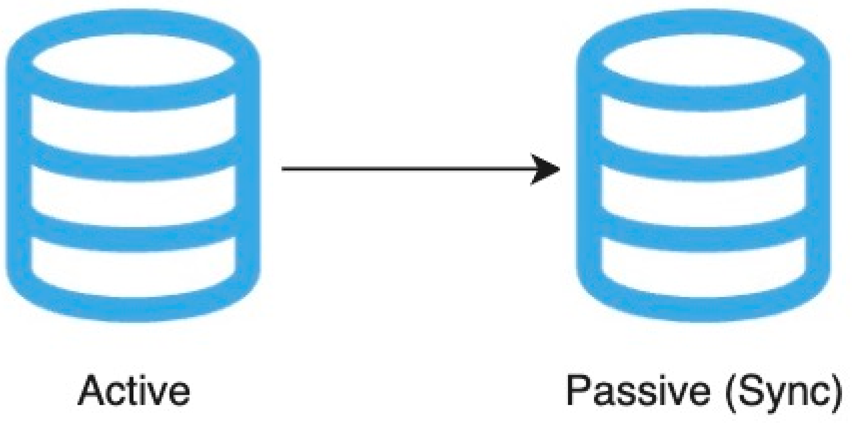
- **Definition**: Maximum **acceptable downtime** after a failure
- **Focus**: **How quickly** can you restore service?
- **Example**:
    - RTO = 1 hour → System must be back online within 1 hour

RPO (Recovery Point Objective)

- **Definition**: Maximum **acceptable data loss** (measured in time)
- **Focus**: **How much data** can you afford to lose?
- **Example**:
    - RPO = 10 minutes → You can tolerate losing 10 minutes of data

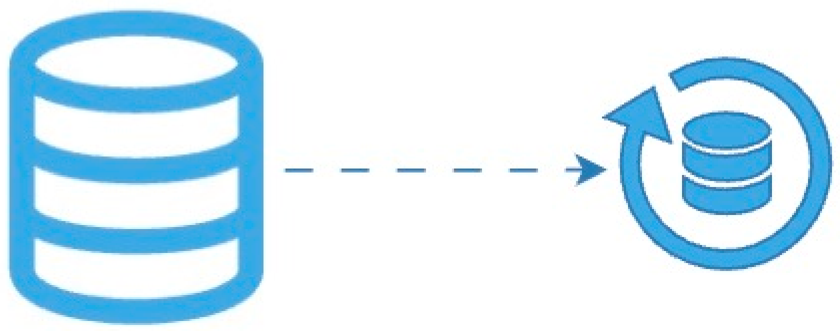
RTO vs RPO

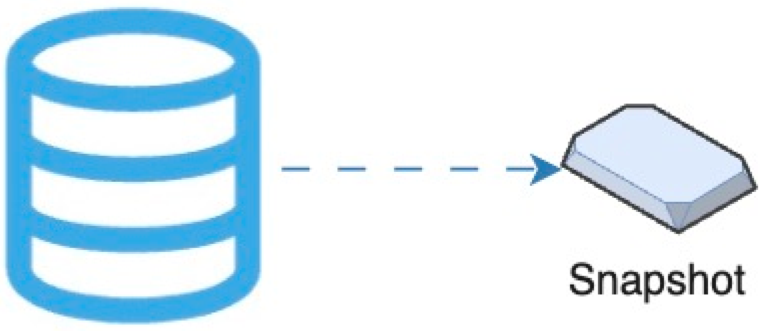
Metric	What It Measures	Example	Focus
RTO	Max downtime allowed	Recover in 1 hour	**Time to recover**
RPO	Max data loss allowed	Lose max 10 minutes of data	**Data protection**



Choosing the Right Strategy

RTO/RPO Requirement	Strategy Example
High RTO + High RPO	Scheduled backups, manual restore
Medium RTO/RPO	Snapshot-based recovery
Low RTO + Low RPO	Active-active replication with automatic failover

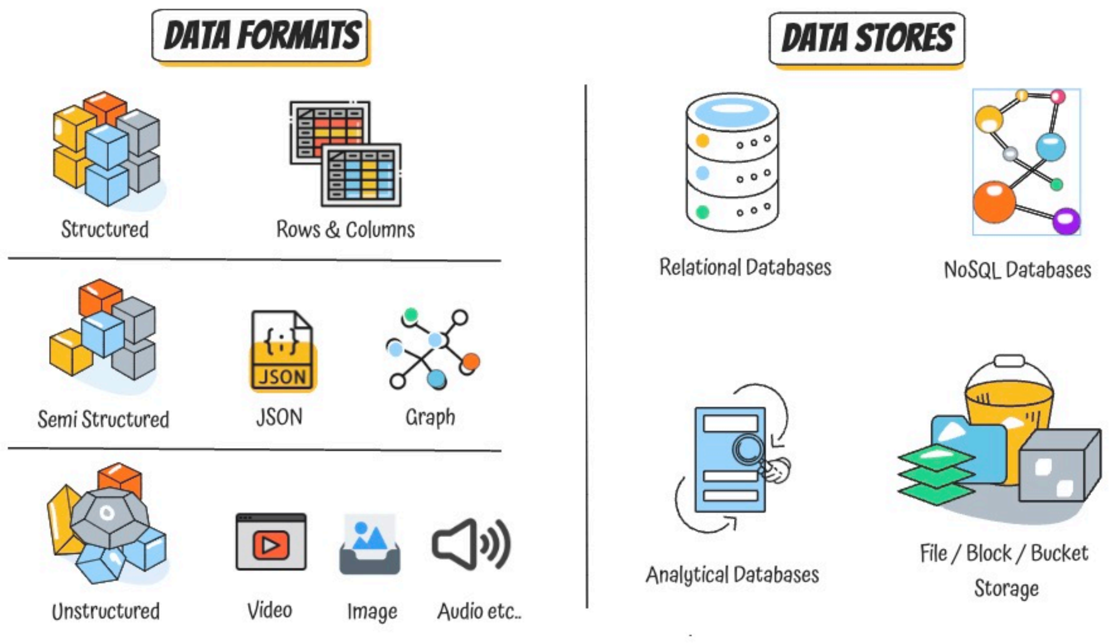




Compare Data Formats vs Data Stores

Types of Data Formats

- **Structured**: Tables, rows, and columns — Typically used by Relational databases (e.g., bank records)
- **Semi-Structured**: JSON, XML, key-value pairs — Typically used by NoSQL databases
- **Unstructured**: Files like images, audio, video, PDFs



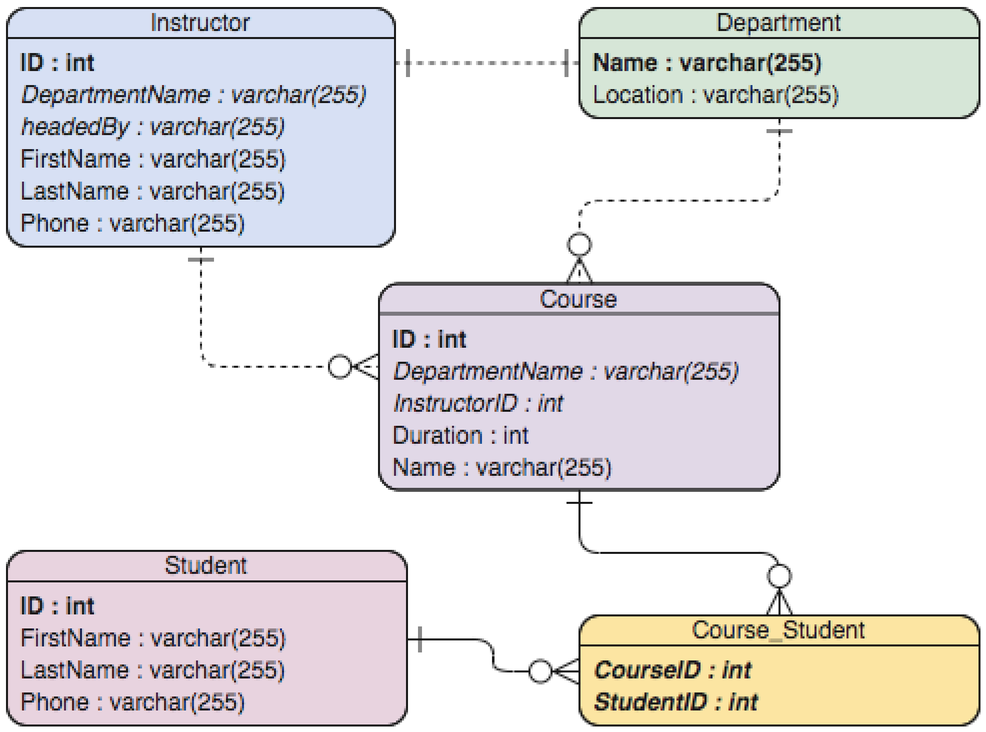
Types of Data Stores

- **Definition**: Services that store, manage, and retrieve different types of data
- **Types**:
  - **Relational Databases** – Structured transactional data
  - **NoSQL Databases** – Flexible document, key-value, or graph data
  - **Analytical Databases** – Petabyte-scale analytics and reporting
  - **Object/Block/File Storage** – Used for backups, media, and other unstructured data

Which Data Format for which Data Store?

- **Structured**: Relational and Analytical
- **Semi-Structured**: NoSQL
- **Unstructured**: Object/Block/File Storage

Where is Structured Data stored? OLAP vs OLTP?



Relational Databases (Structured)

- **Tables with Rows & Columns**: Fixed schema, strict relationships
- **Used in**
  - **OLTP**: Fast reads/writes for high-volume transactions
  - **OLAP**: Analytical queries over massive data (stored in columnar format)

OLTP (Online Transaction Processing):

- Small, frequent transactions – Ex: Transfer money
- **Examples**:
    - Banking system – money transfers, balance checks
    - E-commerce – order placement, payment updates
    - Reservation systems – booking tickets, seat updates

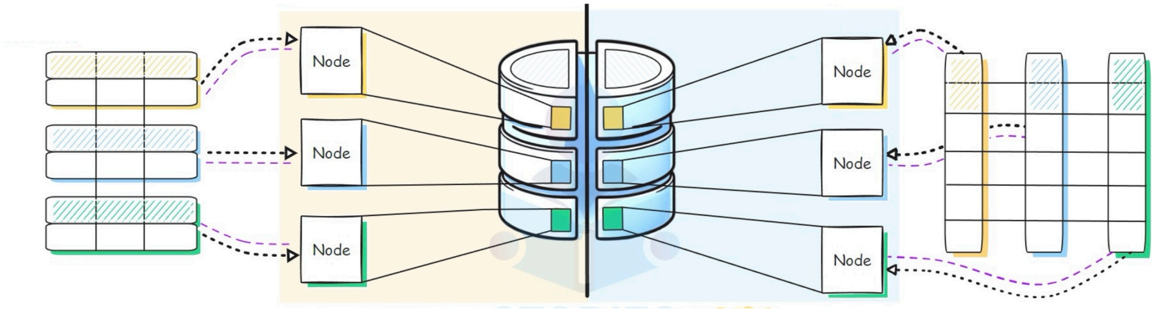
OLAP (Online Analytical Processing):

- Run complex queries on historical data
- **Examples**:
    - Sales trends by region over months
    - Customer behavior analytics
    - Financial reporting and dashboards

OLTP vs OLAP

Aspect	**OLTP**	**OLAP**
Purpose	Day-to-day transactions	Complex data analysis
Query Type	Short, simple, real-time queries	Long-running, analytical queries

Aspect	OLTP	OLAP
Examples	Bank transfers, orders, bookings	Sales reports, user behavior analysis
Terminology	Transactional Databases	Analytical Databases, Data Warehouse
Data Structure	Row-based	Column-based
Typical Architecture	One Large Node with standby (Some modern databases are Distributed)	Distributed



Relational Databases in the Cloud

Type	AWS	Google Cloud	Azure
OLTP	Amazon Relational Database Service, Amazon Aurora	Cloud SQL, Cloud Spanner	Azure Database for MySQL and Azure Database for PostgreSQL, Azure SQL Database
OLAP	Amazon Redshift	BigQuery	Azure Synapse Analytics

Global vs Regional Relational OLTP Databases

Feature	Global Database	Regional Database
Definition	A database deployed across multiple regions, offering low-latency global access and automatic replication	A database deployed in a single region; all data and compute stay local

Feature	Global Database	Regional Database
Availability	Very High – Resilient to regional failures	High – Limited to availability zones in one region
Cost	Higher due to multi-region storage and replication	Lower, with cost tied to a single region
Example – AWS	Amazon Aurora Global Database	Amazon RDS
Example – Azure	Azure SQL with geo-replication (creates a continuously synchronized, readable secondary database)	Azure SQL Database (single region), Azure Database (MySQL/PostgreSQL/..)
Example – Google Cloud	Cloud Spanner (multi-region instance)	Cloud SQL

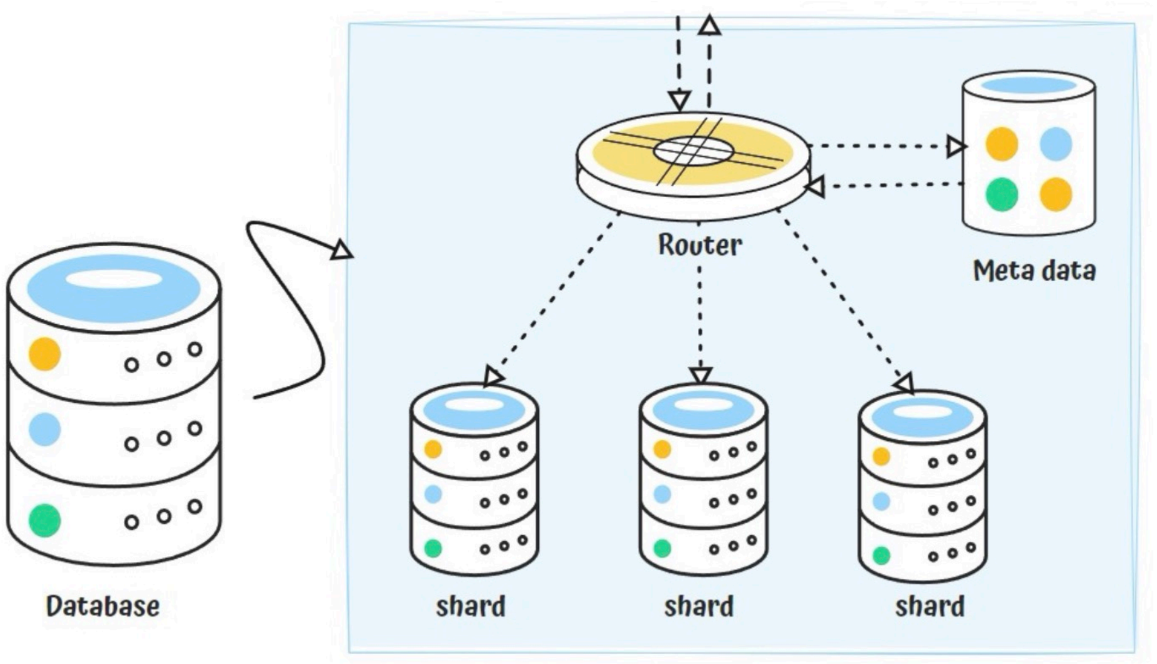
Where is Semi-Structured Data stored? (NoSQL Databases)

Why Semi-Structured Data?

- **Scenario**: Imagine building a product catalog or user profile where each record has different fields — some users have a Twitter handle, others do not
- **Need**: A flexible format that can evolve with your application without changing the database schema every time
- **Definition**: Data with some structure but not rigid like relational tables
- **Examples**: JSON documents, key-value pairs, graphs
- **Use Cases**: Profiles, product catalogs, ..

NoSQL Databases are used to store Semi-Structured Data

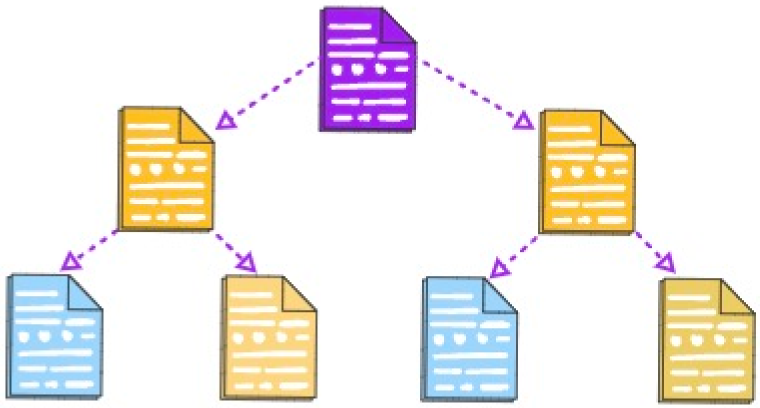
- **NoSQL = Not Only SQL**: Flexible schema, high performance, and horizontal scalability
- **Designed For**: Massive scale and rapid changes in data format
- **Adaptable**: App controls the schema instead of the database



Important NoSQL Database Types

- 1: Document Databases
- 2: Key-Value Databases
- 3: Graph Databases
- 4: Column-Family Databases

1: Document Databases

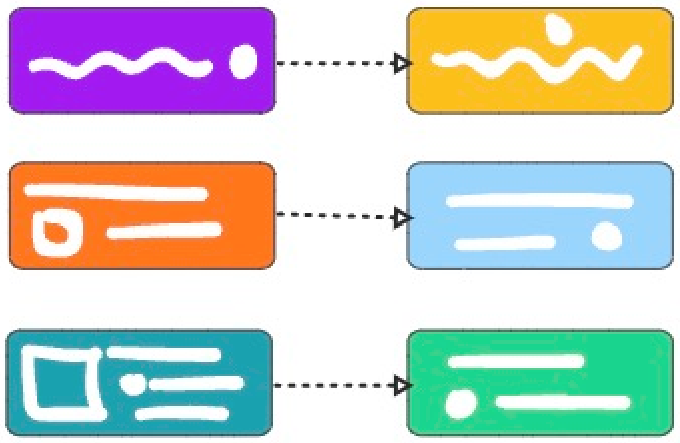


- Data stored as JSON-like documents
- Each document has a unique key
- Structure can vary from document to document
- **Use Cases**: Shopping cart, user profile, product catalog
- **Managed Services**: Amazon DynamoDB, Amazon DocumentDB , Azure Cosmos DB (SQL API), Google Cloud Firestore

```
"user_profiles": [
  {
    "id": 101,
    "name": "Alice",
    "email": "alice@example.com",
    "twitter": "@alice_dev"
  },
  {
    "id": 102,
    "name": "Bob",
    "email": "bob@example.com"
    // Bob has no Twitter handle
  }
]

"product_catalog": [
  {
    "id": "A1",
    "name": "Smartphone",
    "brand": "BrandX",
    "camera_specs": "12MP",
    "battery_life": "10h"
  },
  {
    "id": "B2",
    "name": "Laptop",
    "brand": "BrandY",
    "ram": "16GB",
    "storage": "512GB SSD"
    // No camera_specs for laptops
  }
]
```

2: Key-Value Databases

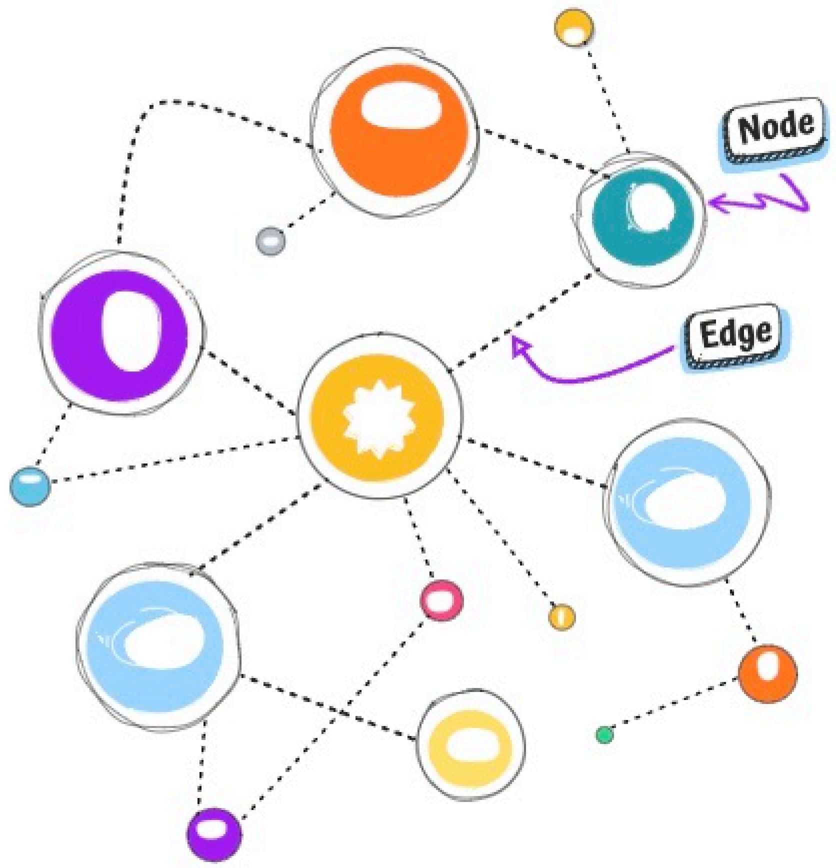


- Data stored as a key and its corresponding value
- Very fast lookups by key
- **Use Cases**: Caching, session management
- **Managed Services**: Amazon DynamoDB, Azure Cosmos DB (Table API), Google Cloud Firestore

```
//session1
{
  "key": "abc123",
  "value": {
    "userId": "u001",
    "loginTime": "2050-07-24T10:00:00Z",
    "role": "admin"
  }
}

//session2
{
  "key": "xyz789",
  "value": {
    "userId": "u002",
    "loginTime": "2050-07-24T10:05:00Z",
    "role": "viewer"
  }
}
```

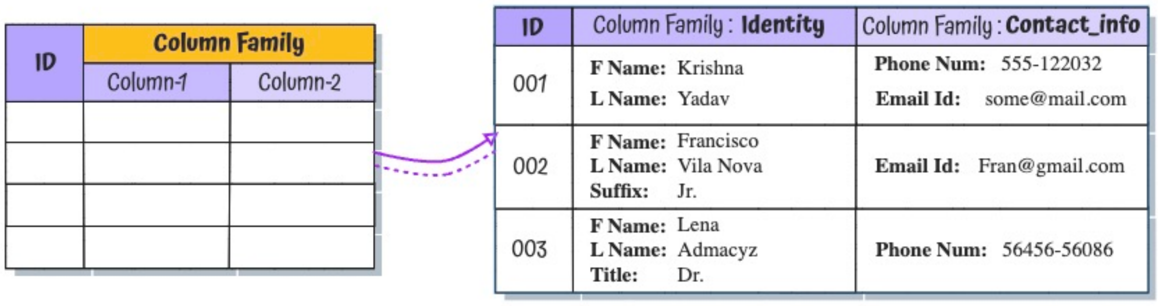
3: Graph Databases



- Data modeled using nodes and relationships (edges)
- Great for capturing and querying complex relationships
- **Use Cases**: Social networks, recommendation engines, fraud detection
- **Managed Services**: Amazon Neptune, Azure Cosmos DB Gremlin API, Spanner Graph

```
{
  "nodes": [
    { "id": "u1", "name": "Ranga" },
    { "id": "u2", "name": "Ravi" },
    { "id": "u3", "name": "John" },
    { "id": "u4", "name": "Sathish" }
  ],
  "edges": [
    { "from": "u1", "to": "u2", "label": "FRIEND" },
    { "from": "u2", "to": "u3", "label": "FRIEND" },
    { "from": "u3", "to": "u4", "label": "FRIEND" },
    { "from": "u4", "to": "u1", "label": "FRIEND" }
  ]
}
```

4: Column-Family Databases



ID	Column Family	
	Column-1	Column-2

ID	Column Family : Identity	Column Family : Contact_info
001	F Name: Krishna L Name: Yadav	Phone Num: 555-122032 Email Id: some@mail.com
002	F Name: Francisco L Name: Vila Nova Suffix: Jr.	Email Id: Fran@gmail.com
003	F Name: Lena L Name: Admacyz Title: Dr.	Phone Num: 56456-56086

- Data stored in rows and columns grouped into families
- Sparse format – rows don't need to have all columns
- **Use Cases**: IoT data, time-series data, analytics
- **Managed Services**: Amazon Keyspaces (Cassandra), Azure Cosmos DB Cassandra API, Google Cloud Bigtable

Example

```
{
  // Unique identifier for the row
  // typically identifies a device, user, or servic
  "rowKey": "device123",

  "columnFamilies": {

    // First column family stores time-based log en
    "logs": {
      // Timestamp as column name, log message as v
      "2050-07-24T10:00:00Z": "Temperature: 32°C",
      "2050-07-24T10:01:00Z": "Temperature: 33°C",
      "2050-07-24T10:02:00Z": "Temperature: 34°C"
    },

    // Second column family stores system statuses
    "status": {
      // Same timestamp as column name, status mess
```

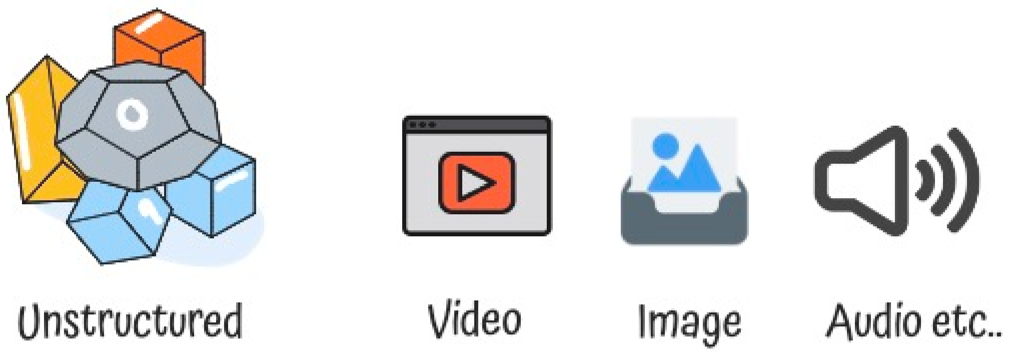
```
            "2050-07-24T10:00:00Z": "OK",
            "2050-07-24T10:01:00Z": "OK",
            "2050-07-24T10:02:00Z": "ALERT: Temp threshol
          }
        }
      }
```

Where is Unstructured Data stored? (File, Block, or Object storage)

Why Unstructured Data?

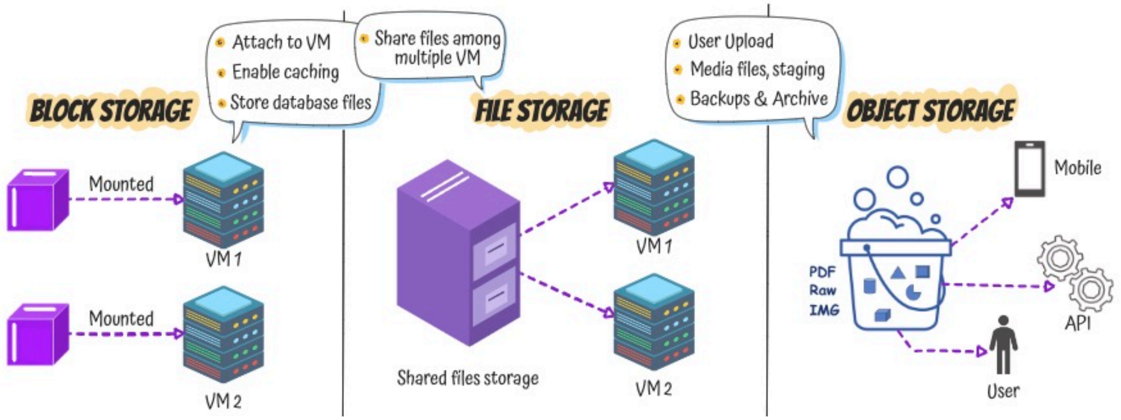
- **Scenario**: Imagine building YouTube — videos, thumbnails, subtitles, and logs. All this content doesn't fit into a table.
- **Need**: A way to store and retrieve large files like videos, images, documents, and logs — without predefined structure.
- **Definition**: Data without a fixed schema or format (e.g., audio, video, PDFs, images, binaries)
- **Examples**: Uploaded files, media content, backup archives, sensor logs

- **Handled Using**: File, block, or object storage based on access pattern and use case

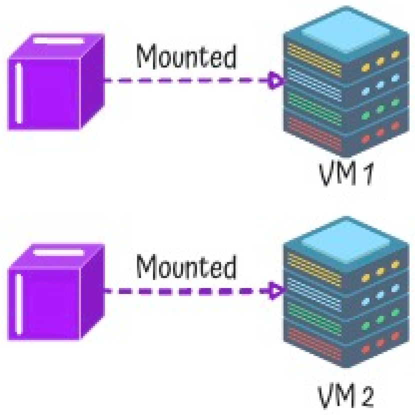


Types of Storage for Unstructured Data

- 1: Block Storage
- 2: File Storage
- 3: Object Storage

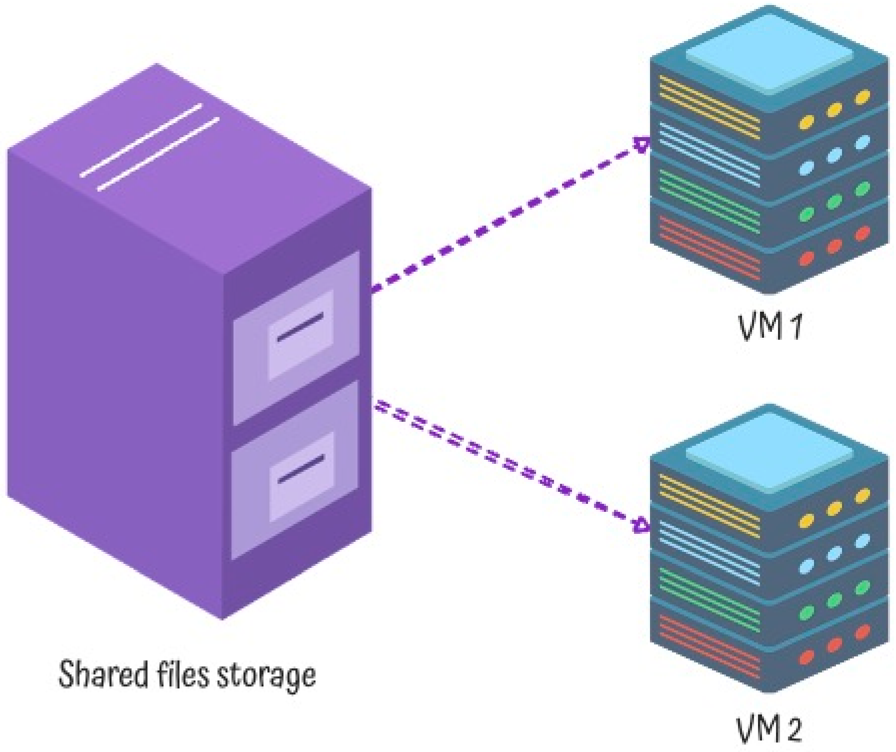


1: Block Storage



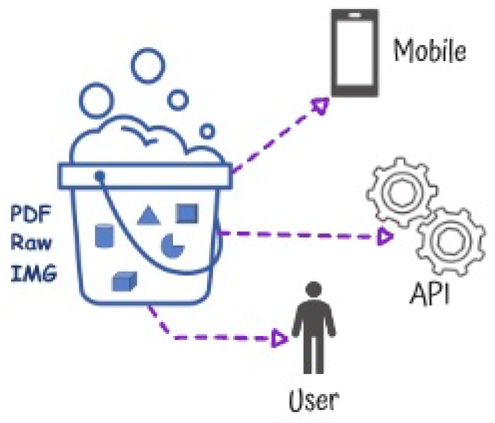
- Low-level storage used like a hard disk
- High performance for structured workloads (e.g., VM disks, databases)
- **Use Cases**: OS disks, DB volumes

2: File Storage



- Shared file systems accessed over network using file paths
- Ideal for applications needing traditional file structure and shared access
- **Use Cases**: Team file shares, CMS systems

3: Object Storage



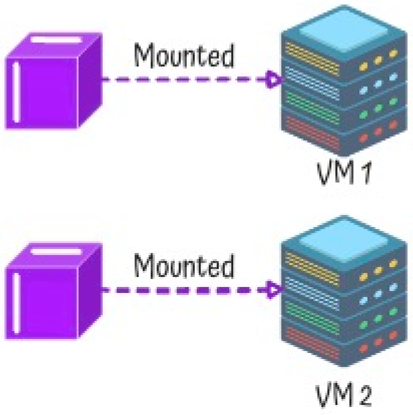
- Data stored as objects (data + metadata + unique ID)
- Accessed using REST APIs — no mounting required
- Scalable, cost-effective, and durable
- **Use Cases**: Media hosting, backups, logs, static websites

What is Block Storage?

Why Block Storage in Cloud?

- **Scenario**: Imagine running a virtual machine or a database in the cloud. You need a reliable, fast disk that behaves like a physical hard drive.

- **Block Storage**: Provides raw storage volumes that can be attached to servers and used just like local disks.
- **Goal**: Attach virtual disks to compute resources like VMs, containers, and databases



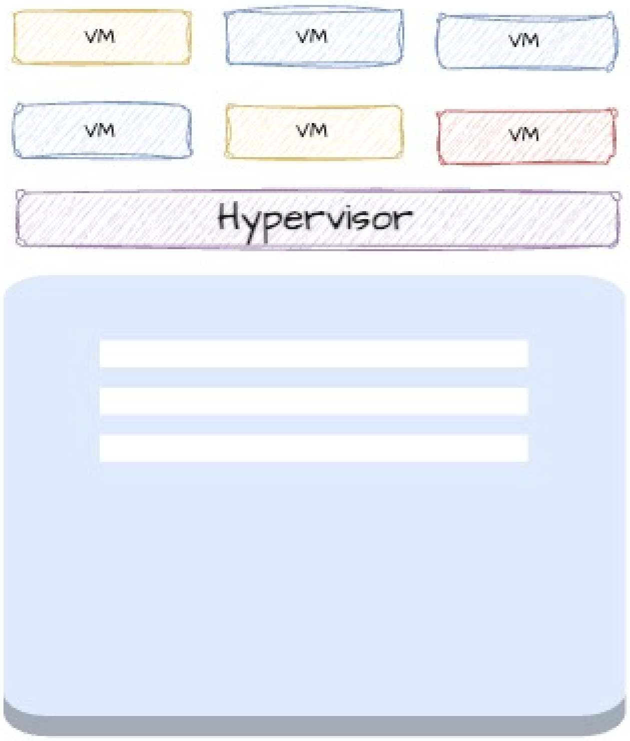
Key Characteristics

- **Raw Volumes**: You format and mount it like a traditional disk
- **Detachable and Reusable**: Can be attached, detached, and re-attached to different servers
- **Persistent**: Data stays intact even if the VM is stopped or restarted

Use Cases

- **Virtual Machines**: OS and data disks for cloud-based servers

- **Databases**: High-speed transactional storage for SQL and NoSQL engines



Choice 1: Types of Block Storage

- **Persistent Block Storage (e.g., Network Attached like EBS)**
    - Stored separately and connected over the network
    - **Retains data** across VM stops, starts, or replacements
    - Ideal for critical data like databases or file systems
- **Temporary Block Storage (e.g., Instance Store)**

    - Physically attached to the VM host
    - Very fast but **data is lost** if VM is terminated
    - Best for temporary data like cache or scratch files

Choice 2: HDD (Hard Disk Drive) vs SSD (Solid State Drive)

- **HDD (Hard Disk Drive)**
    - **Transactional Performance**: Lower – slower for frequent reads/writes
    - **Throughput**: High – good for sequential access of large files
    - **Strength**: Best for large, sequential data processing
    - **Use Cases**:
        - Big data workloads
        - Log processing
        - Backup or archival
    - **Cost**: Lower – budget-friendly
- **SSD (Solid State Drive)**
    - **Transactional Performance**: High – excellent for fast, frequent access
    - **Throughput**: High – handles both small and large data well
    - **Strength**: Great for small, random and sequential I/O
    - **Use Cases**:

        - Databases
        - Web servers
        - Operating system volumes
    - **Cost**: Higher – but justified for high performance

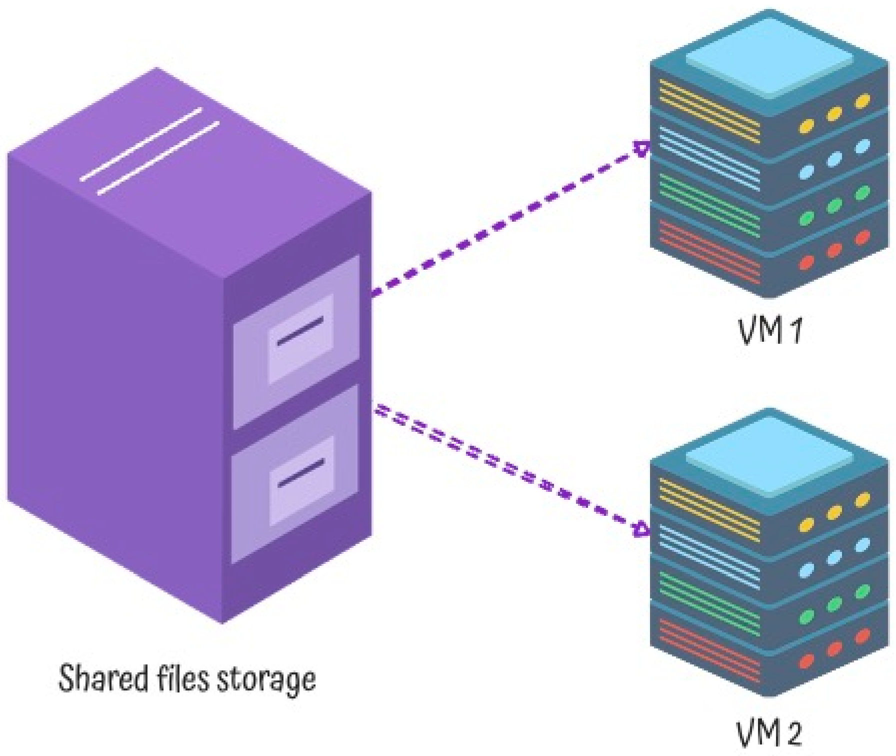
Cloud Managed Services for Block Storage

- **AWS**: Amazon EBS, Instance Store
- **Azure**: Azure Managed Disks, Temporary Disks
- **Google Cloud**: Persistent Disks, Local SSDs

What is File Storage?

Why File Storage?

- **Scenario**: Imagine a team working on shared documents, code files, or project reports. Everyone needs access to the same files, organized in folders.
- **File Storage**: A storage system that organizes data in a familiar folder and file format, accessible over a shared network.



What is File Storage?

- **Definition**: A way to store data in *hierarchical* structure – folders and files
- **Goal**: Share files easily between users or systems
- **Access Method**: Files accessed using standard protocols like NFS or SMB
    - **NFS**: A protocol designed to share files over a network in **Linux/Unix** systems

    - **SMB**: A protocol used mainly in **Windows environments** to share files

How File Storage Works

- **Mounted Volumes**: File storage is mounted like a network drive
- **Shared Access**: Multiple users or servers can read/write files
- **Folders & Files**: Organize data just like on your laptop

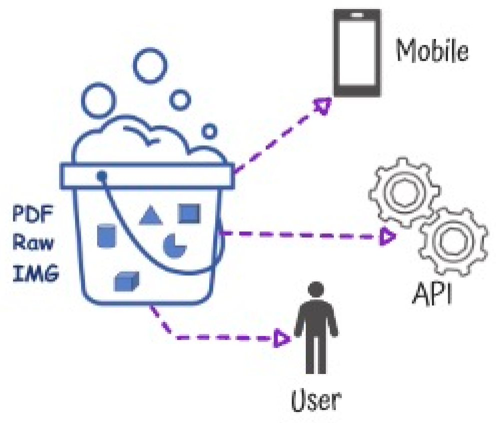
Benefits of File Storage

- **Easy to Use**: Familiar structure – folders, paths, extensions
- **Shared Access**: Ideal for collaboration
- **Reliable**: Supports backups, snapshots, and versioning

Cloud Managed Services for File Storage

- **AWS**: Amazon EFS (Elastic File System), Amazon FSx
- **Azure**: Azure Files
- **Google Cloud**: Filestore

What is Object Storage?



Why Object Storage in Cloud?

- **Scenario**: Imagine millions of users uploading photos, videos, and documents through a website or mobile app — and accessing them anytime, from anywhere. Traditional file systems struggle to scale, manage access, or serve global traffic efficiently.
- **Object Storage**: Designed to store and retrieve large volumes of unstructured data (like media and backups) with high durability, scalability, and global accessibility.
- **Goal**: Store and retrieve any type of file at scale using a simple API

Key Characteristics

- **Flat Structure**: No folders — everything is stored in a bucket with a unique key

- **Scalable**: Can handle billions of files without performance loss
- **Durable**: Data is automatically replicated across multiple zones or regions
- **Access via HTTP APIs**: Easy to integrate with applications, websites, and mobile apps

Use Cases

- **Backup and Archiving**: Reliable, long-term storage
- **Web Content**: Store and serve images, videos, documents
- **Big Data and Analytics**: Ingest large files for processing
- **Application Storage**: Store logs, exports, reports

Choice 1: Versioning

- **Purpose**: Keep multiple versions of an object to protect against accidental deletes or overwrites
- **Use Case**: Restore a previous version of a file or track changes over time

Choice 2: Storage Tiers/Storage Class

- **Purpose**: Store data in the most cost-effective tier based on how often it is accessed

- **Balance Performance and Price**: Pay more for speed when needed, save more when data is rarely used
- **Example Tiers**
  - **Standard or Hot**: For frequently accessed data
  - **Infrequent Access or Cool**: For data accessed less often
  - **Archive**: For long-term storage with slower retrieval

Choice 3: Lifecycle Rules

- **Purpose**: Automatically transition or delete objects based on age or other characteristics
- **Use Case**: Move old files to cheaper storage (e.g., archive) or delete them after a set period to save costs

Cloud Managed Services for Object Storage

- **AWS**: Amazon S3 (Standard, IA, Glacier, , ..)
- **Azure**: Azure Blob Storage (Hot, Cool, Archive , ..)
- **Google Cloud**: Cloud Storage (Standard, Nearline, Coldline, Archive, ..)

Interesting Thing to Know

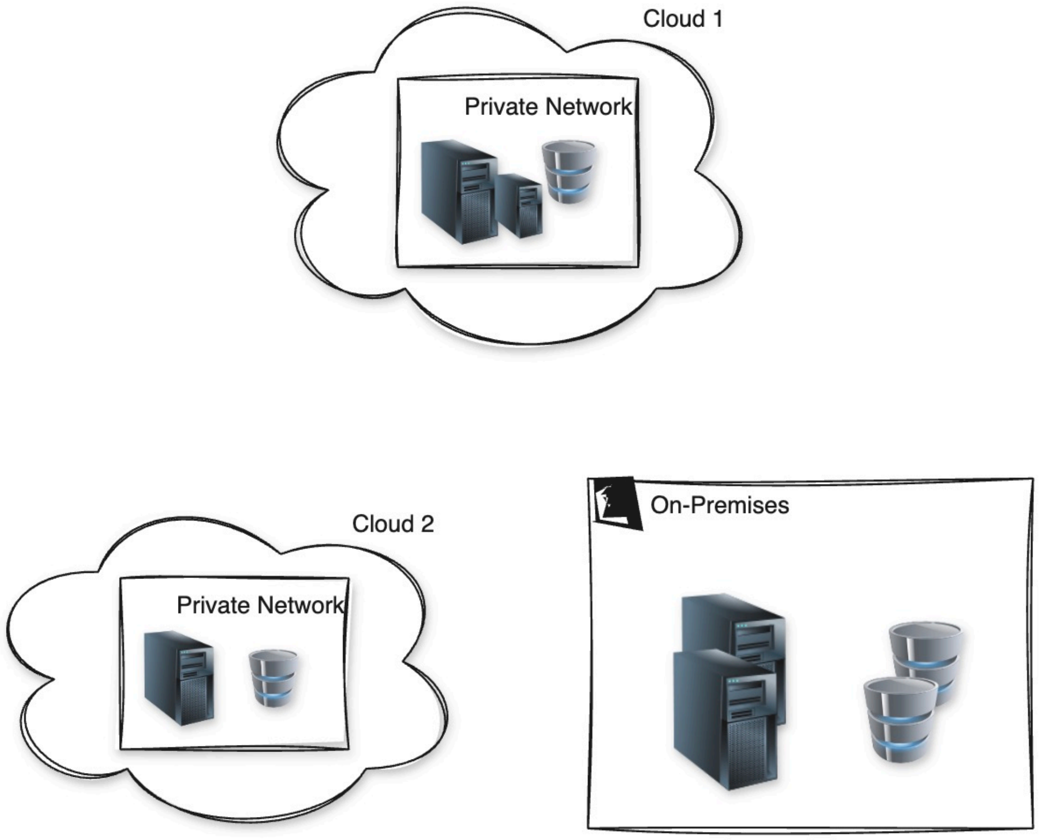
- **Amazon Glacier Service**: Amazon Glacier was launched as a stand-alone service for long-term storage with very low cost.

- **Optimized for Archiving**: Great for backups, compliance data, and archived content that can wait minutes or hours to be retrieved.
- **Overtime Integrated into S3**: Part of Amazon S3 as a storage class
- **Simple Management**: Use S3 interface to store data in Glacier – no need to manage a new service.
- **All Clouds Offer Archive Storage as a Storage Class/Tier**: AWS (Glacier), Azure (Archive), GCP (Coldline, Archive)

What is Hybrid Storage?

Why Hybrid Storage?

- **Scenario**: Imagine storing large datasets which need to be accessed on-premises — some frequently accessed, some rarely touched. Keeping all of it in the cloud may be slow. Keeping it all on-prem may limit scalability.
- **Hybrid Storage**: Combines **on-premises** and **cloud storage** to balance cost, speed, and control.



What is Hybrid Storage?

- **Definition**: A storage solution that bridges **local (on-premises)** storage with **cloud storage**
- **Goal**: Enable seamless data access and movement between on-prem and cloud environments
- **Use Cases**:

    - Gradual cloud migration
    - Burst storage for peak workloads
    - Archive and backup to cloud

Benefits of Hybrid Storage

- **Scalability**: Cloud extends your capacity without new hardware
- **Cost Efficiency**: Store only hot data locally, cold data in the cloud
- **Performance**: Local access for critical data
- **Data Protection**: Cloud backup improves disaster recovery

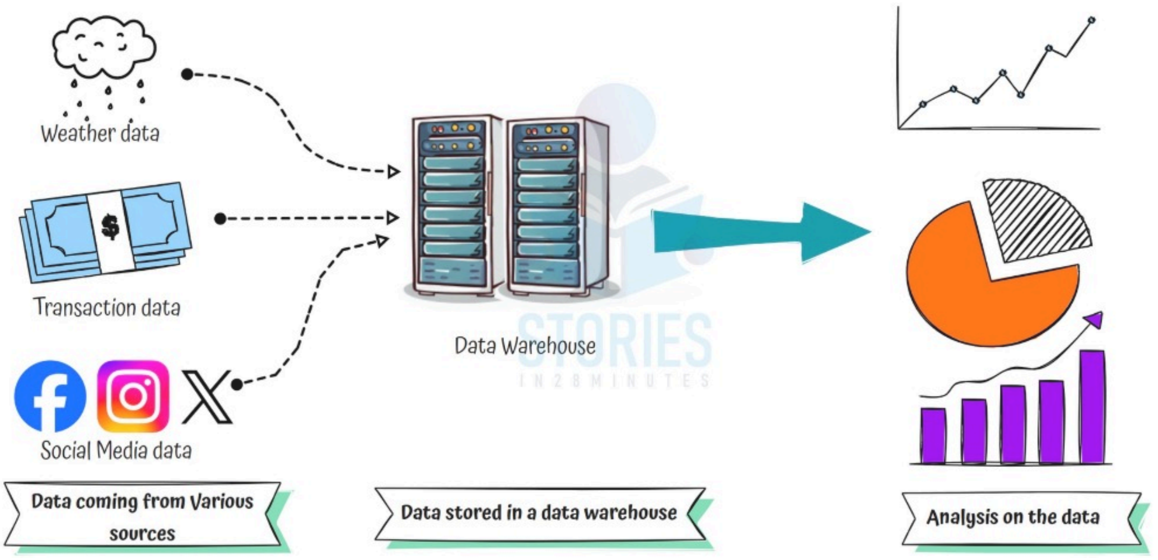
Cloud Managed Services for Hybrid Storage

- **AWS**: AWS Storage Gateway
- **Azure**: Azure File Sync
- **Google Cloud**: Google Cloud Filestore (with Hybrid Connectivity)

What is the need for Data Analytics?

Why Data Analytics?

- **Scenario**: You have tons of raw data — purchases, transactions, sensor readings. But without analysis, it's just noise.
- **Data Analytics**: The process of analyzing raw data to extract useful insights
- **Goal**: Make data-driven decisions to improve business outcomes



Data Sources Can Include

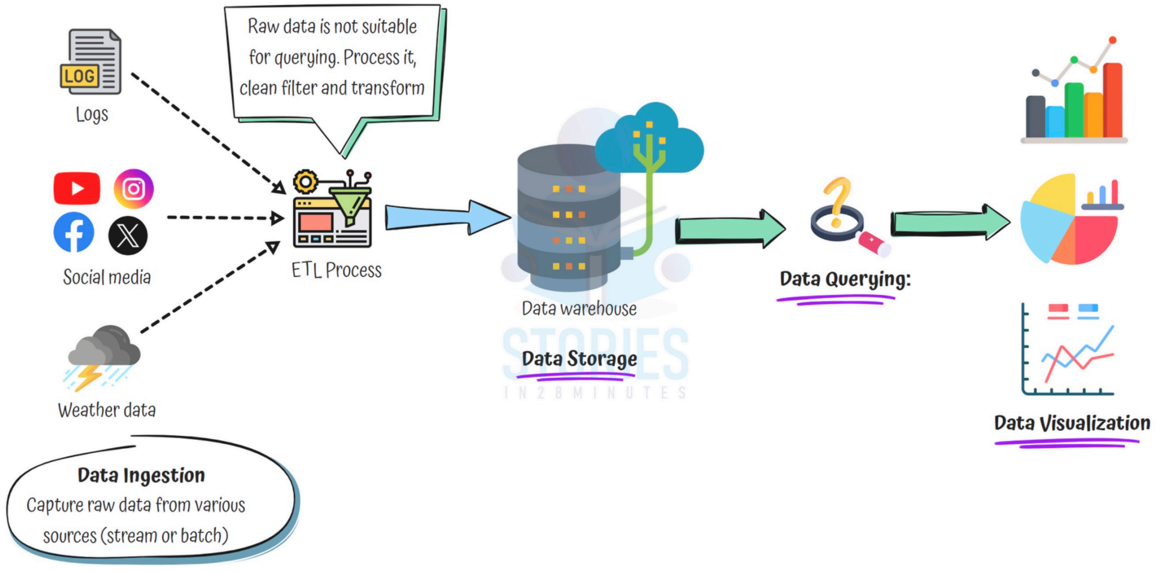
- **Transactions**: Purchases, payments, logs
- **Sensor & IoT Data**: Temperature, pressure, GPS
- **External Feeds**: Weather, stock prices, social media
- **Internal Systems**: CRM, HR, finance systems

Key Benefits of Data Analytics

- **Uncover Trends**: Understand customer behavior, market shifts, and product performance
- **Identify Weaknesses**: Spot bottlenecks, inefficiencies, or risks early
- **Improve Outcomes**: Enhance efficiency, customer satisfaction, and profitability

Why Data Analytics Workflow?

- **Scenario**: You collect tons of raw data from multiple sources. But raw data is not useful until it's cleaned, processed, and visualized.
- **Workflow**: Follows a clear step-by-step path from raw data to business insights.


Logs
Raw data is not suitable for querying. Process it, clean filter and transform
Social media
ETL Process
Data warehouse
Data Storage
Data Querying:
Weather data
Data Visualization
Data Ingestion
Capture raw data from various sources (stream or batch)
STORIES IN 28 MINUTES

Data Ingestion

- **Goal**: Collect raw data from multiple sources
- **Sources**: Websites, IoT sensors, apps, logs, transactions
- **Modes**:
  - **Batch**: Data loaded periodically
  - **Stream**: Data ingested in real-time (e.g. user clicks, weather sensors)

Data Processing

- **Goal**: Make data usable for analysis
- **Clean**: Remove duplicates and errors

- **Filter**: Eliminate irrelevant or outlier data
- **Transform**: Convert to a consistent format or structure
- **Aggregate**: Combine data for summaries or insights

Data Storage

- **Where**: Store in a **data warehouse**
- **Goal**: Centralize data for easy access and future analysis

Data Querying

- **What**: Run SQL-like queries to analyze trends, identify patterns

Data Visualization

- **Why**: Charts and dashboards make insights easier to understand
- **Impact**: Helps leadership spot trends, outliers, and make informed decisions

Cloud Managed Services for Data Analytics

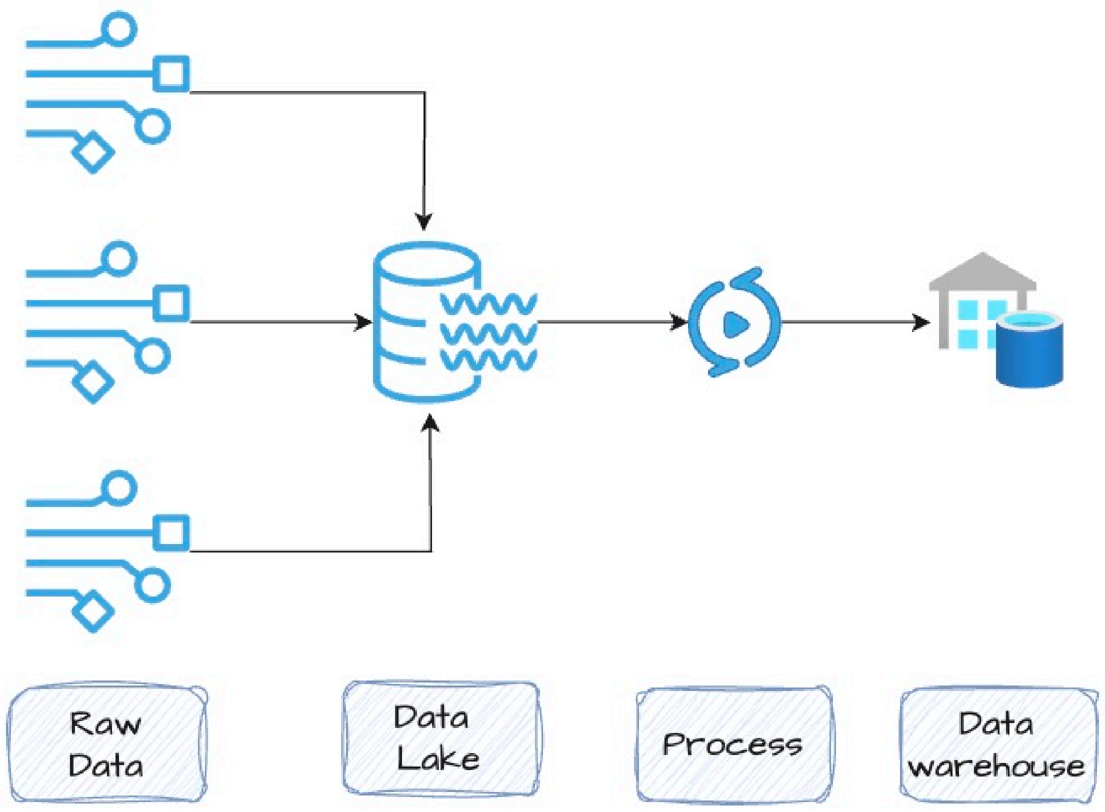
Stage	AWS	Azure	Google Cloud
Streaming Ingestion	Amazon Kinesis	Azure Event Hubs	Pub/Sub
Data Processing (ETL / Data Prep)	AWS Glue	Azure Data Factory	Dataflow, Dataprep
Data Warehouse and Querying	Amazon Redshift	Azure Synapse Analytics	BigQuery
Data Visualization	Amazon QuickSight	Power BI	Looker

Compare Data Warehouse vs Data Lake

The 3Vs of Big Data

- **Volume**: Massive datasets — from terabytes to petabytes to exabytes

- **Variety**: Mix of structured (tables), semi-structured (JSON), and unstructured (videos, logs)
- **Velocity**: Speed of data arrival — batch (hourly/daily) or real-time (streams)



What if the Data We're NOT Capturing Becomes Valuable Later?

- Businesses may **miss future insights** if they only store processed data
- **Storing raw data now** gives the flexibility to run AI, ML, or new analytics later
- A **Data Lake** solves this — store everything today, use what you need tomorrow

Data Warehouse vs Data Lake

- **Data Warehouse**
  - Stores processed, structured data optimized for fast SQL queries
  - Used for business intelligence, dashboards, and reporting
  - **Examples**: Teradata, Amazon Redshift, Google BigQuery, Azure Synapse Analytics
- **Data Lake**
  - Stores raw, unprocessed data — compressed and cost-efficient
  - Can handle any format (CSV, JSON, images, audio, logs, etc.)
  - Supports on-demand exploration, AI/ML, and analytics workflows
  - Built on object storage
  - **Examples**: Amazon S3, Google Cloud Storage, Azure Data Lake Storage Gen2

How They Work Together

- **Data Lake** holds everything — raw logs, events, images, clickstreams, ...
- **Data Warehouse** pulls from the lake — after processing
- **Modern Tools**: Services like Google BigQuery, Azure Synapse Analytics, and Amazon Athena can **query data directly from the data lake** — no need to move or duplicate

Cloud Managed Services for Big Data Storage and Analytics

- **AWS**:
    - Storage: Amazon S3
    - Warehouse: Amazon Redshift
    - Query-over-lake: Amazon Athena
- **Azure**:
    - Storage: Azure Data Lake Storage Gen2
    - Warehouse: Azure Synapse Analytics
    - Query-over-lake: Synapse Serverless SQL
- **Google Cloud**:
    - Storage: Google Cloud Storage
    - Warehouse: BigQuery

- Query-over-lake: BigQuery External Tables

Previous
Cloud Fundamentals

Next
Asynchronous Communication in the Cloud