in28minutes

Getting Started with Data Formats and Data Stores

What are Different Data Formats?

Structured: Well organized data in spreadsheets or tables

- **Example**: Bank account details

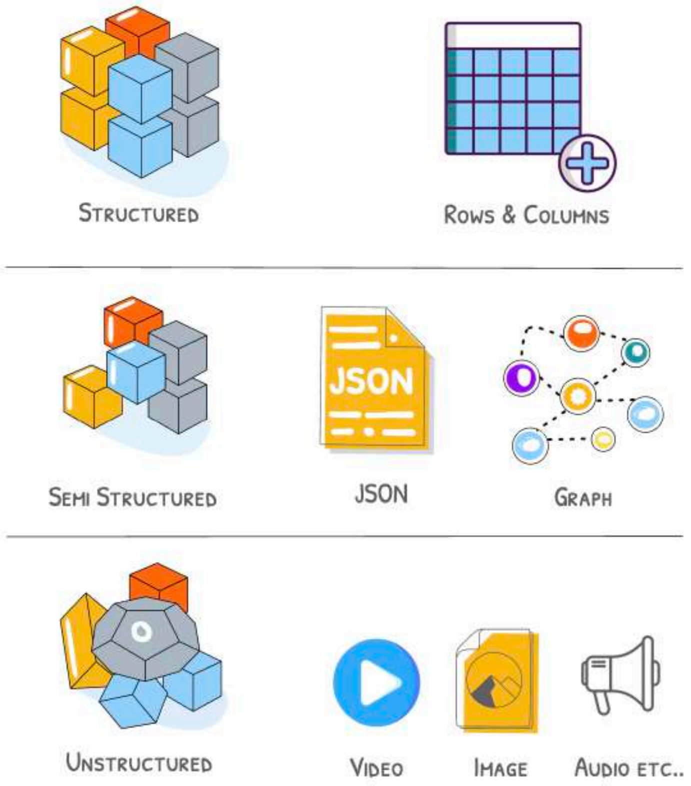
Semi-Structured: Data with some structure but providing flexibility

- **Examples**: JSON, XML, key-value pairs

Unstructured: Raw files

- **Examples**: Images, videos, PDFs, audio files

Key Insight: Format determines how you store data

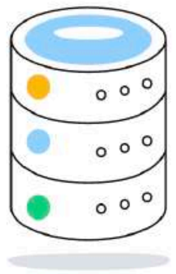


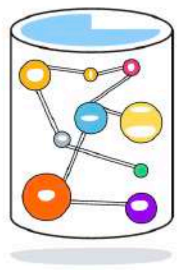
What are Different Types of Data Stores?

Data Stores: Services that store, manage, and retrieve data

- **Relational Databases**: Designed for structured data with tables and well-defined columns
- **NoSQL Databases**: Designed for flexible, semi-structured data (JSON, XML, Key Value Pairs, etc)
- **Object / Block / File Storage**: Used for unstructured data - Images, videos, PDFs, audio files

Decision: Choose right store based on data type and use case


Relational Databases


NoSQL Databases


Analytical Databases


File / Block / Bucket Storage

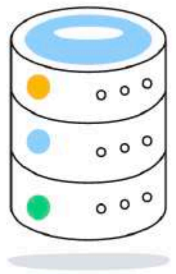
Let's Get on a Data Journey!

Terminology: We introduced quite a few terminology in this step

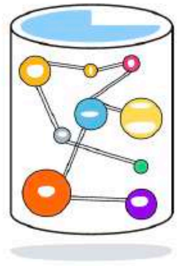
Data Formats: Structured, Semi Structured and Unstructured

Data Stores: Relational Databases, NoSQL Databases and Object/Block/File Storage

How about a Journey: To understand them in depth with a lot of examples



RELATIONAL DATABASES



NOSQL DATABASES

ANALYTICAL DATABASES

FILE / BLOCK / BUCKET STORAGE

in28minutes

Getting Started with Structured Data

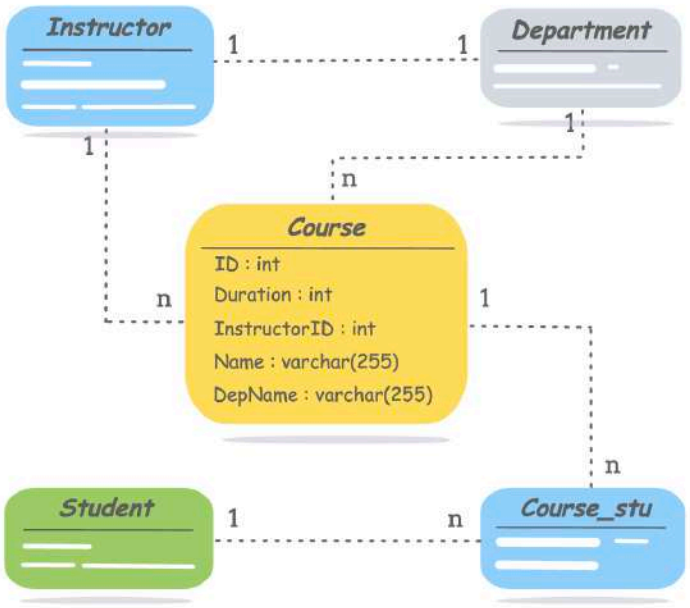
Exploring Structured Data – Our Goal

Basics: What is Structured Data?

Storage: Why Relational Databases are the right fit?

Use Cases: Where is Structured Data used?

- **OLTP**: Transactional Use Cases
- **OLAP**: Analytical Use Cases
- **Comparison**: OLTP vs OLAP



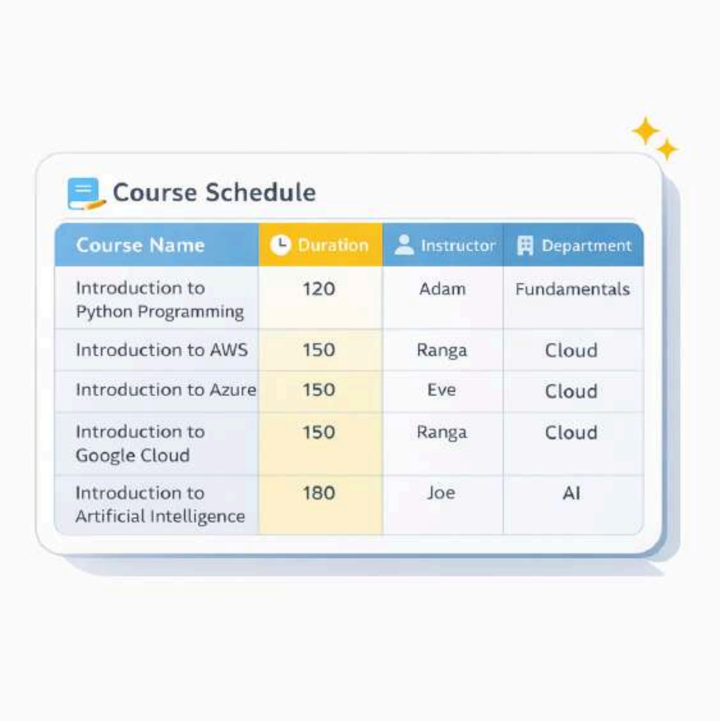
What is Structured Data?

Definition: Highly organized data that fits a rigid, predefined structure

Analogy: Think of a simple, neat Excel spreadsheet

- **Real-Life Example**: A list of courses with all details
- **Another Example**: A school attendance sheet with attendance marked for the current month

Why It's Useful: You can find things quickly because everything is in the right place



Course Schedule

Course Name	Duration	Instructor	Department
Introduction to Python Programming	120	Adam	Fundamentals
Introduction to AWS	150	Ranga	Cloud
Introduction to Azure	150	Eve	Cloud
Introduction to Google Cloud	150	Ranga	Cloud
Introduction to Artificial Intelligence	180	Joe	AI

Where is Structured Data Stored?

Relational Databases: Data Stored in Tables

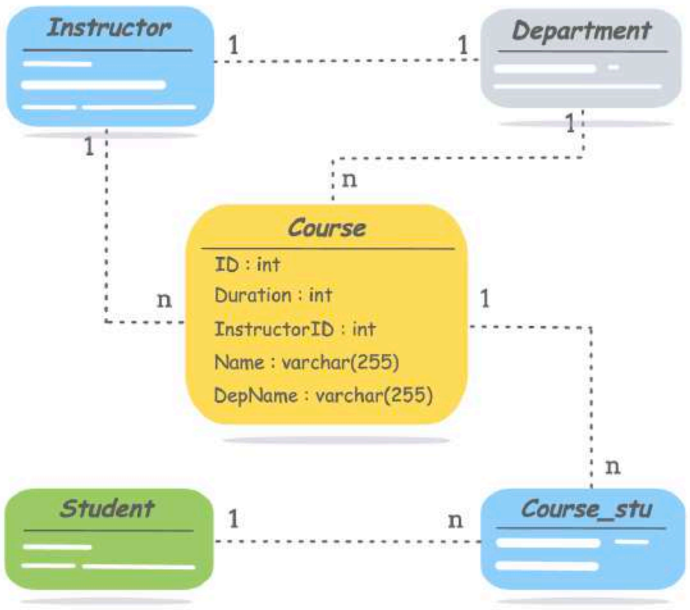
Well Defined Structure: Tables have multiple columns with clearly defined data types

- **Uniform Data:** All rows have identical column structure

Relationships: Tables have relationships

- **Example:** Course has an Instructor, Course belongs to Department

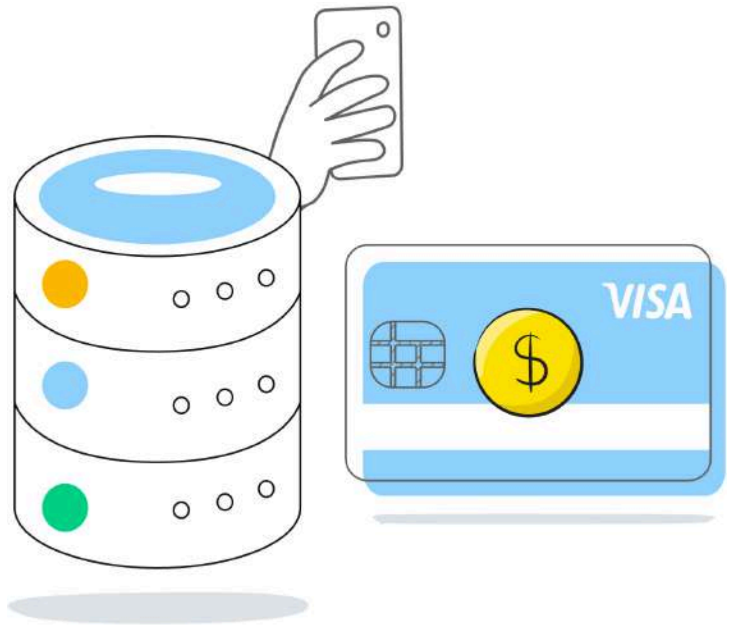
Common Examples: Oracle, MySQL, PostgreSQL



What is OLTP (Online Transaction Processing)?

OLTP Applications: Process thousands of transactions every second

- **Transaction:** Small, discrete unit of work
- **Examples:**
  - **Banking:** Making transfers
  - **E-commerce:** Placing orders
  - **Reservations:** Booking tickets



Design: OLTP Design Goal

- **Fast Response:** Fast read and write transactions
- **High Concurrency:** Hundreds of thousands of simultaneous users

What is OLAP (Online Analytical Processing)?

OLAP Applications: Applications analyzing huge volumes of data

Use Cases: Data warehousing, business intelligence, and reporting

- **Example**: Calculating insurance premiums based on 100 years of data
- **More Examples**: Customer behavior analytics, Financial reporting and dashboards

Goal: Transform data into actionable insights

OLTP vs OLAP – Examples

Domain	OLTP Example	OLAP Example
Banking	Money transfers, balance checks	**Customer Insights** – Identify high-value customers with frequent transactions. **Risk Analysis** – Detect unusual transfer patterns for fraud prevention.
E-commerce	Order placement, payment updates	**Sales Trends** – Analyze top-selling products by season.
Reservations	Booking tickets, seat updates	**Demand Forecasting** – Identify peak booking periods for dynamic pricing. **Customer Preferences** – Analyze popular routes and destinations

Exploring Scenarios: Structured Data

Scenario	Solution
Where is Structured Data Stored?	Relational Databases
OLAP or OLTP: Instantaneous inventory updates for an e-commerce site	OLTP
OLAP or OLTP: Analyzing 5 years of sales history to predict next quarter's demand	OLAP Data Warehouse
OLAP or OLTP: Processing a high volume of small customer transactions	OLTP
OLAP or OLTP: Creating a dashboard showing regional sales performance over the last week	OLAP Data Warehouse

in28minutes

Getting Started with Semi Structured Data

Real World Data is NOT Always Structured

Scenario: Consider user profile data (or a product catalog)

- **Variations:** One user has a `twitter` handle; another does not
- **Missing Fields:** Some records might have `courses` but lack `email`
- **Multiple Values:** Some fields might have multiple values (`courses`)

Semi-Structured: Data has some structure but is often incomplete with lot of variations

```
{
    "id": 101,
    "name": "Alice",
    "email": "alice@example.com",
    "twitter": "@alice_dev"
}

{
    "id": 102,
    "name": "Bob",
    "email": "bob@example.com"
    // Bob has no Twitter handle
}

{
    "id": 103,
    "name": "Ranga",
    "courses": ["Google Cloud", "AWS",
              "Azure","Generative AI"]
    // No Email, No Twitter
}
```

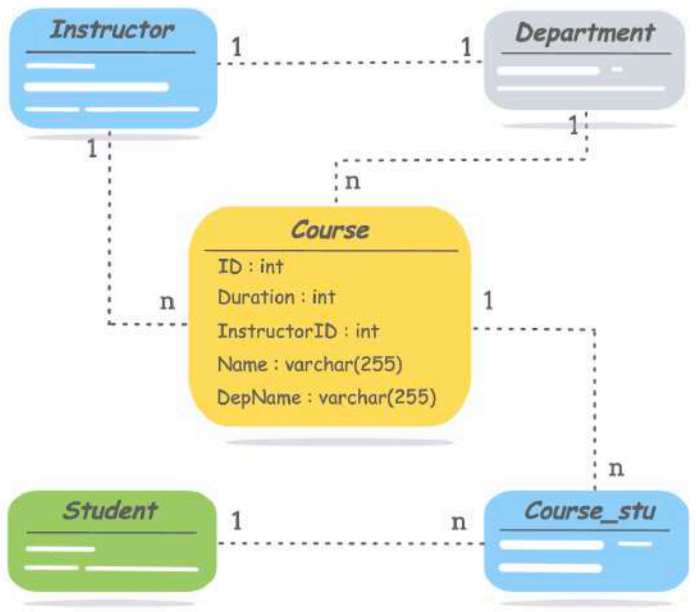
The Relational Database Rigidity Problem

The Challenge: Relational databases (SQL) need fixed, rigid schema

Upfront Schema: Define all tables, columns, and data types before adding data

- **Schema Change**: Making a change (like adding a `twitter` or `courses` column) needs schema update

Result: Difficult to store semi structured data in relational databases



Need – Databases To Store Flexible Data

Goal: Store Semi-Structured Data

- **No Rigid Schema**: Allows you to add or remove fields from records easily
- **Support High-Speed Development**: Allows data format to **evolve** with the application

Common Use Cases:

- **User Profiles**: Different users have different attributes
- **Product Catalogs**: Products have highly varied attributes (e.g., shoe vs laptop)

```
{
    "id": 101,
    "name": "Alice",
    "email": "alice@example.com",
    "twitter": "@alice_dev"
}

{
    "id": 102,
    "name": "Bob",
    "email": "bob@example.com"
    // Bob has no Twitter handle
}

{
    "id": 103,
    "name": "Ranga",
    "courses": ["Google Cloud", "AWS",
             "Azure","Generative AI"]
    // No Email, No Twitter
}
```

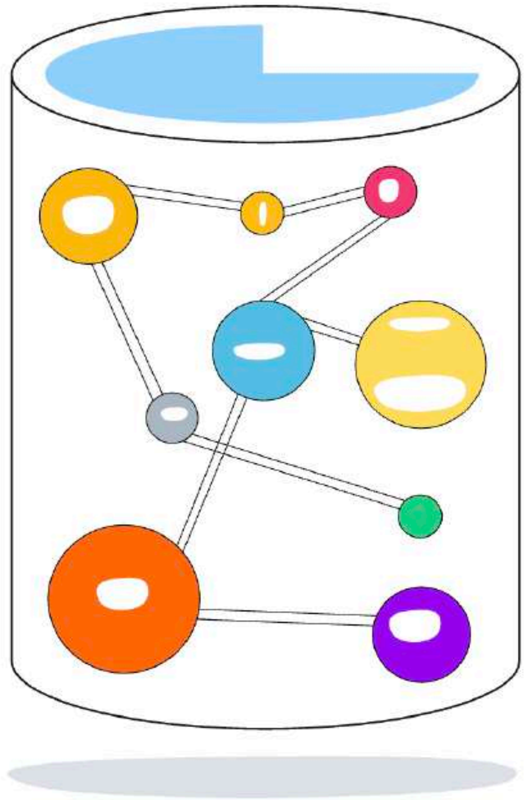
What are NoSQL Databases?

NoSQL Databases: Used to Store Semi-Structured Data

- **NoSQL**: Not Only SQL
- **Adaptable**: App controls schema (NOT database)
- **Designed For**: Huge scale

Types: 4 Key Types

- Document Databases
- Key-Value Databases
- Graph Databases
- Column-Family Databases



What are Document Databases?

Scenario: You need to store flexible, semi-structured data like user profile or a product catalog

- **Document Databases**: Store data as JSON-like documents, each identified by a **unique key**
- **Key Feature**: Optimized for fast lookup with keys

Managed Services:

- **AWS** – Amazon DynamoDB, Amazon DocumentDB
- **Azure** – Azure Cosmos DB (SQL API)
- **Google Cloud** – Cloud Firestore

```
{
   "customerId": "99999999",
   "firstName": "Ranga",
   "lastName": "Ranga",
   "address": {  //Child Object - `{}`
      "number": "505",
      "street": "Main Street",
      "city": "Hyderabad"
   },
   "socialProfiles": [ //Array - `[]`
      {
         "name": "twitter",
         "username": "@in28minutes"
      },
      {
         "name": "linkedin",
         "username": "rangaraokaranam"
      }
   ]
}
```

Why JSON is Popular in Document Databases

Human-Readable: Easy to read

Flexible Schema: No fixed structure

- **Easier Evolution:** Add or remove fields anytime
- **Everything Together:** Related data (customer + address + profiles) stored as one document
- **Nested Data Support:** Capture complex relationships within one document

Programming Friendly: Natively supported in JS, Python, Java, etc.

- **Fewer Joins, Faster Access:** Simplifies queries and improves performance

```
{
   "customerId": "99999999",
   "firstName": "Ranga",
   "lastName": "Ranga",
   "address": {  //Child Object - `{}`
      "number": "505",
      "street": "Main Street",
      "city": "Hyderabad"
   },
   "socialProfiles": [ //Array - `[]`
      {
         "name": "twitter",
         "username": "@in28minutes"
      },
      {
         "name": "linkedin",
         "username": "rangaraokaranam"
      }
   ]
}
```

What are Key-Value Databases?

Requirement: Extremely fast access to data using a unique key

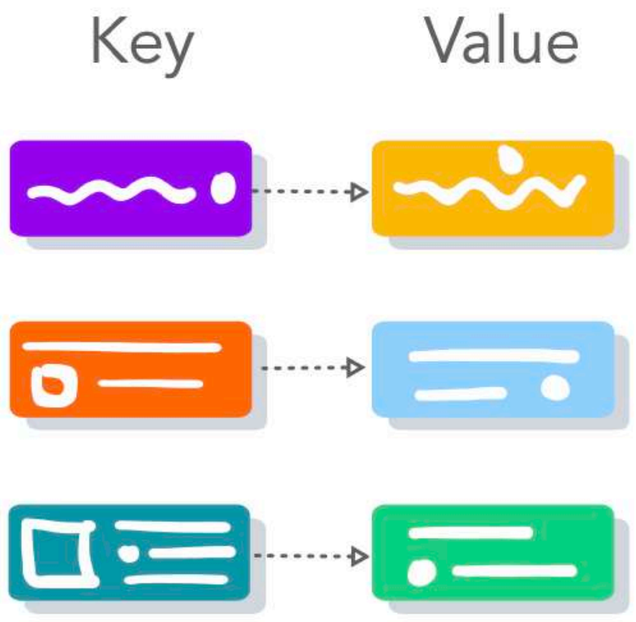
- **Use Cases**: Caching Data, Session Management

Key-Value Databases: Store data as key–value pairs

- **Fast Lookup**: Great for high-performance, low-latency lookups by key (NOT optimized for lookup by values)

Managed Services:

- **AWS** – Amazon DynamoDB
- **Azure** – Azure Cosmos DB (Table API)
- **Google Cloud** – Cloud Firestore



What are Key-Value Databases? Example

```
//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"
   }
}
```

What are Graph Databases?

Scenario: You need to model and query complex relationships between entities

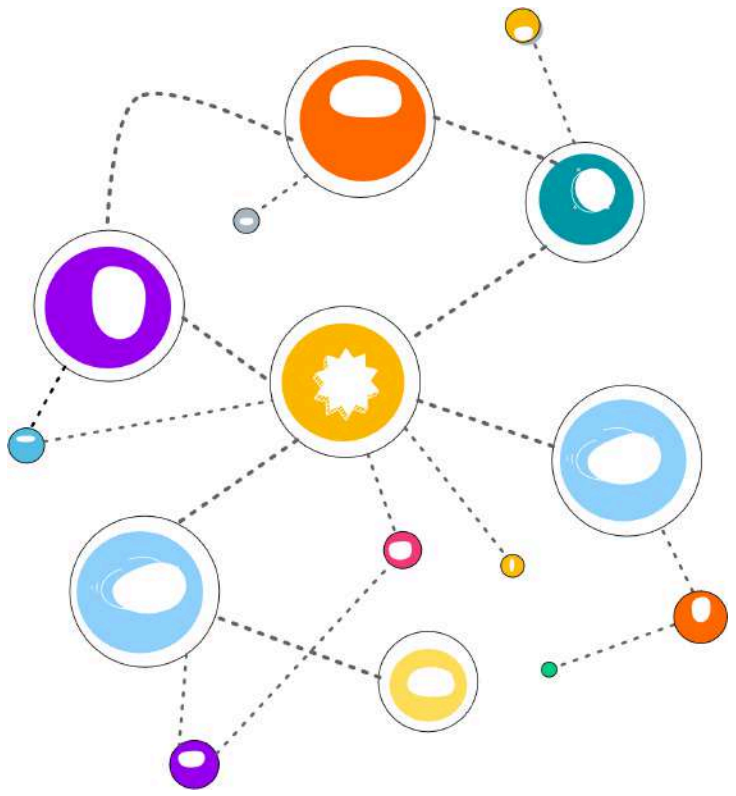
- **Use Cases**: Social networks, recommendation engines, fraud detection

Graph Databases: Store data as **nodes** (entities) and **edges** (relationships)

- **Answer**: Who knows whom?, What connects them?

Managed Services:

- **AWS** – Amazon Neptune
- **Azure** – Azure Cosmos DB (Gremlin API)
- **Google Cloud** – Spanner Graph



What are Graph Databases? Example

```json
{
  "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": "u1", "label": "FRIEND" },
    { "from": "u3", "to": "u4", "label": "FRIEND" },
    { "from": "u4", "to": "u2", "label": "FRIEND" }
  ]
}
```

What Is Time-Series Data?

Definition: Data recorded over time, where each entry has a **timestamp**

Example: IoT device reporting temperature and status

Pattern: Continuous stream of values (often every second or minute)

Challenge: Data Volume (Millions of devices → billions of time-stamped entries)


```
{
  "rowKey": "device123",
  "logs": {
     "2050-07-24T10:00:00Z": "32°C",
     "2050-07-24T10:01:00Z": "33°C",
     "2050-07-24T10:02:00Z": "34°C"
  },
  "status": {
    "2050-07-24T10:00:00Z": "OK",
    "2050-07-24T10:01:00Z": "OK",
    "2050-07-24T10:02:00Z": "ALERT"
  }
}
```

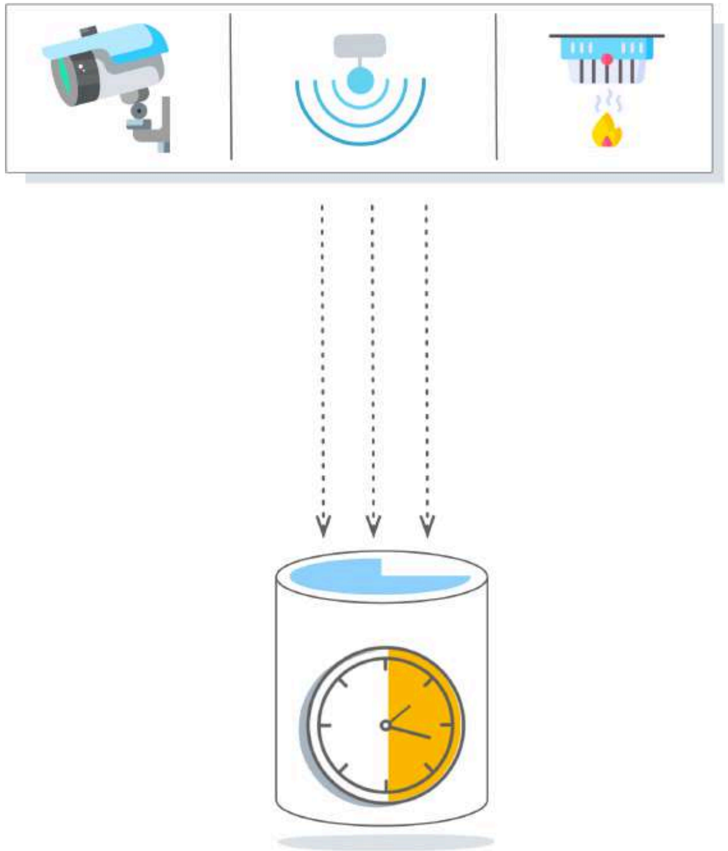
Why Traditional Databases Struggle With Time-Series Data

Relational Databases: Typically store each reading as a single row

- Example: One row for each timestamp

Problems: A few problems

- **Problem 1 – Too Many Rows:** Millions of devices × thousands of readings
- **Problem 2 – Slow Queries:** Read last hour/day of data (Scanning millions of rows becomes slow)
- **Conclusion:** Relational databases are not built for massive, fast, time-indexed workloads



Why Column-Family Databases Fit Time-Series Data

High Write Throughput: Designed for continuous, high-frequency inserts

Flexible Layout: Add new fields (like `status`) without schema migrations

Row Per Device: Each device (e.g., `device123`) becomes one row

- **Column Family**: Stores multiple readings
- **Reading “last 5 min"**: Becomes a fast scan

Result: Ideal for IoT telemetry, metrics, trading updates, and logs

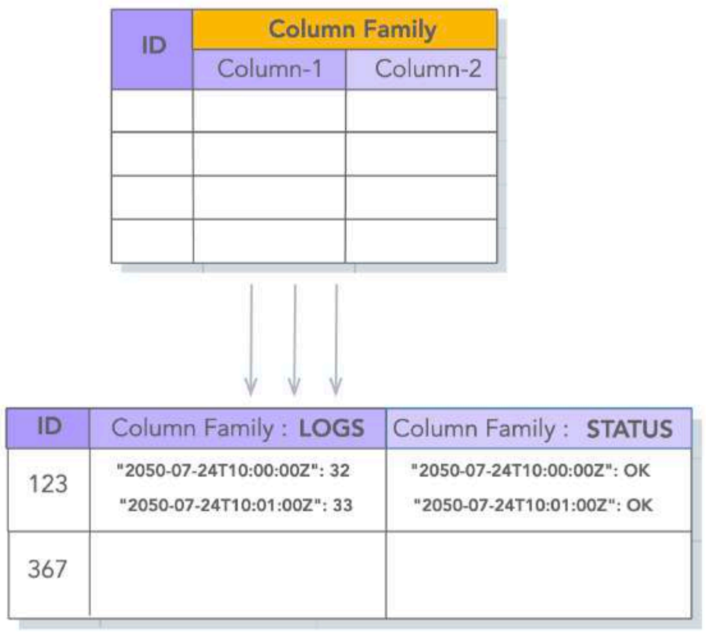
```
//device_id, timestamp, temperature
//device123, 2050-07-24T10:00:00Z,32
//device123,2050-07-24T10:01:00Z,33
//device123,2050-07-24T10:02:00Z,38

{
   "rowKey": "device123",
   "logs": {
      "2050-07-24T10:00:00Z": "32°C",
      "2050-07-24T10:01:00Z": "33°C",
      "2050-07-24T10:02:00Z": "38°C",
   },
   //Flexible
   "status": {
      "2050-07-24T10:00:00Z": "OK",
      "2050-07-24T10:01:00Z": "OK",
      "2050-07-24T10:02:00Z": "ALERT",
   }
}
`
```

What are Column-Family Databases?

Scenario: You need to handle large volumes of time series efficiently

- **Definition:** Store data in rows and flexible columns - **column families**
- **Sparse structure** – Values NOT needed for all columns
- **Use Cases:** IoT streams, time-series data, financial data - stock prices etc



Managed Services:

- **AWS** – Amazon Keyspaces (for Apache Cassandra)
- **Azure** – Azure Cosmos DB (Cassandra API)
- **Google Cloud** – Cloud Bigtable

What are Column-Family Databases? Example

```
{
   // Unique identifier -  identifies a device, user, or service
   "rowKey": "device123",

   "columnFamilies": {

      // First column family stores time-based log entries
      "logs": {
         // Timestamp as column name, log message as value
         "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 message as value
         "2050-07-24T10:00:00Z": "OK",
         "2050-07-24T10:01:00Z": "OK",
         "2050-07-24T10:02:00Z": "ALERT: Temp threshold exceeded"
      }
   }
}
```

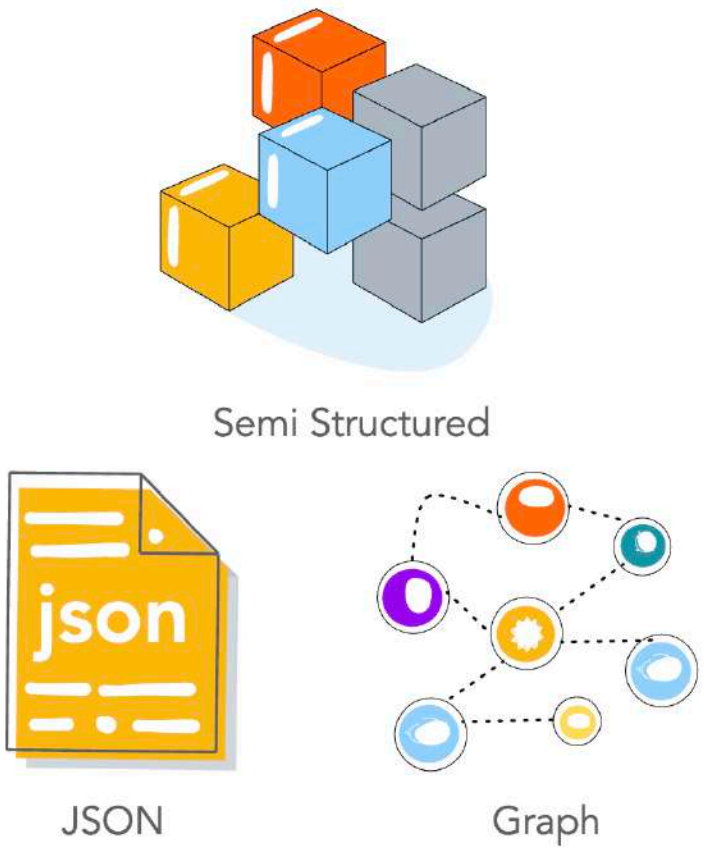
Summary – Semi-Structured Data & NoSQL

Semi-Structured Data: Flexible data formats

- Ideal for evolving applications

NoSQL DBs: Store Semi-Structured Data

- Flexible schemas, high scalability/performance
- **App controls schema**: NOT the database
- **Types:**
  - **Document** → JSON documents, user profiles, product catalogs
  - **Key-Value** → Extremely fast lookup - caching, session management
  - **Graph** → Relationships, social networks, recommendation engines
  - **Column-Family** → Time-series, IoT streams



Semi-Structured Data - Scenarios

Scenario	Best Choice
Storing "Friends of Friends" connections	Graph Database
Storing millions of IoT sensor readings per second	Column-Family / Wide-Column Database
Social Network: "People you may know" feature	Graph Database
Storing shopping cart session data	Key-Value Database
Storing product catalog with different attributes per product	Document Database
IoT: Time-series data from smart meters	Column-Family / Wide-Column Database

Getting Started with Unstructured Data

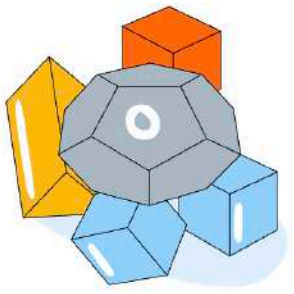
Why Unstructured Data?

Scenario: Building YouTube (videos, thumbnails, and logs don't fit in tables)

- **The Need**: Store and retrieve large files without predefined structures

Unstructured Data: Files like audio, video, PDFs, images, and binaries

- **Examples**: Media content, backup archives, and sensor logs
- **Storage Options**: Stored in Block, File, or Object storage based on use case



Where is Unstructured Data Stored?

Scenario	Storage Type
Store operating system files or application data on a virtual disk and attach it with a VM	**Block Storage** (Amazon EBS, Azure Disks, Google Persistent Disk)
Setup a common file share to share common files with multiple users or applications	**File Storage** (Amazon EFS, Azure Files, Google Filestore)
Upload and access media files (videos, thumbnails, ..) using REST APIs without mounting to VM	**Object Storage** (Amazon S3, Azure Blob Storage, Google Cloud Storage)

What is Block Storage?

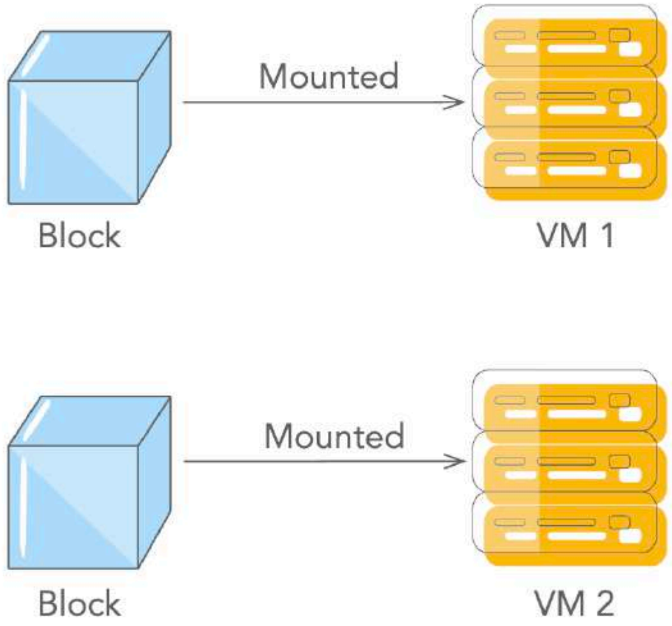
Scenario: Running a Custom Database

- **Requirement**: Fast storage for storing data like physical hard drive

Block Storage: Raw storage volumes attached to VMs

- **Detachable**: Can move volumes between different VMs
- **Persistent**: Data remains intact even if VM stops or restarts

Use Cases: Boot disks, data disks, and high-speed database storage



Boot Disk vs Data Disk

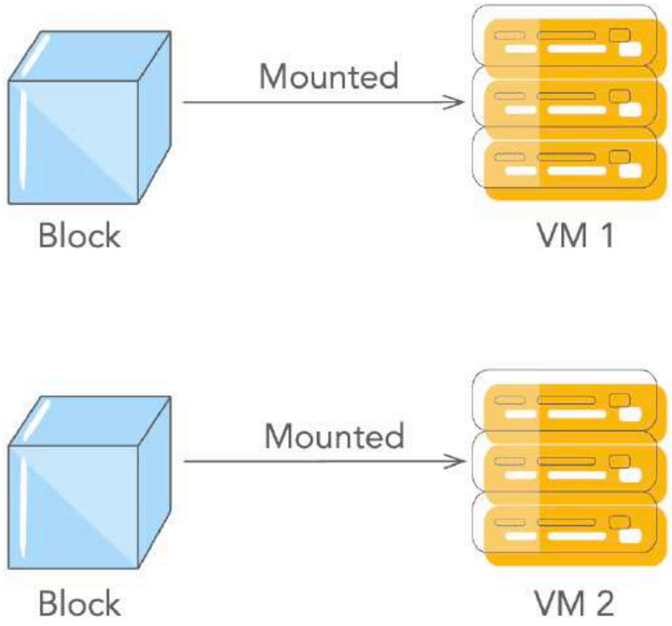
Boot Disk: Contains Operating System (OS) and startup files

- **Purpose**: Used to **boot** the virtual machine
- **Automatically Created**: When you launch a VM instance

Data Disk: Used to store application data, logs, or databases

- **Use Case**: Storing persistent data beyond VM lifecycle

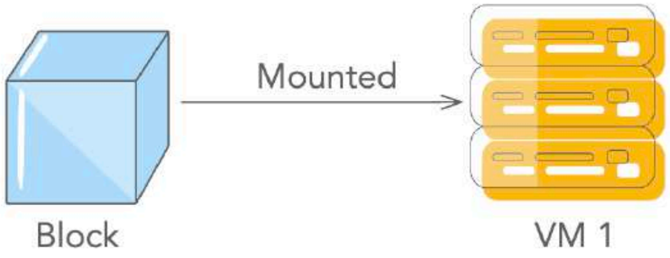
Separation: Best practice to keep OS and Data on different disks



What are Types of Block Storage?

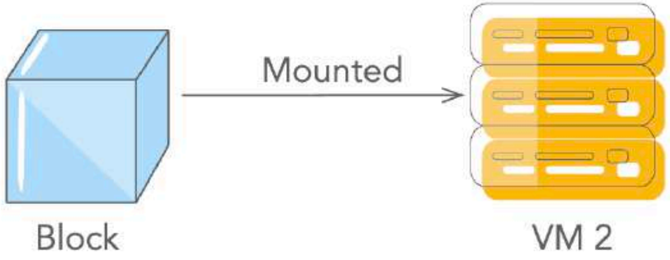
Persistent Block Storage: Network-attached

- **Retains data**: Even if VM is stopped or replaced
- **Ideal for**: Databases, logs, and application data
- **Lifecycle**: Independent of the VM



Ephemeral Block Storage: Physically attached to the VM

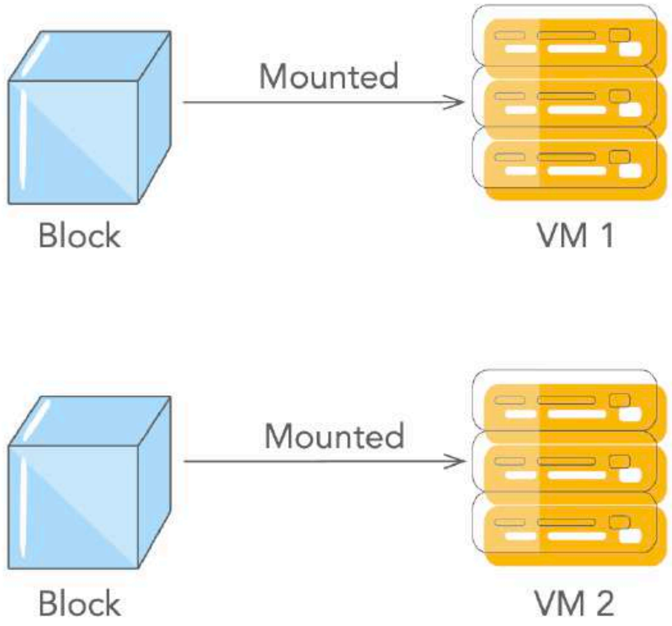
- **Very high performance**: Low latency and high IOPS
- **Temporary data**: Lost when the VM is terminated
- **Ideal for**: Caching, temp files
- **Lifecycle tied** to VM instance: Cannot be detached



What is Persistent Block Storage?

Key Features: A few key features

- **Flexible Lifecycle:** Detach and attach with VMs
- **Provisioned capacity:** Choose capacity and resize when needed - while attached to a VM
- **Performance Scaling:** Performance typically scales with volume size
- **Choice of Regional or Zonal replication:** Get higher availability and durability
- **Snapshots Support:** For regular backups



Use Case: Databases, critical applications, or reusable disks

Persistent vs Ephemeral Block Storage

Feature	Persistent Block Storage	Ephemeral Block Storage
Attachment	Network-attached	Physically attached to the VM server
Lifecycle	Independent; remains after VM deletion	Tied to the VM; deleted on termination
Data Safety	Highly durable; survives stops and reboots	Lost when VM terminates
Performance	Moderate/High (Network latency applies)	Very High (Local access)
Performance Scaling	Often scales with volume size	Fixed
Snapshots	Supported (Point-in-time backups)	Not supported
Use Cases	Databases, Boot disks, Critical app data	Caches, Scratch space, Temporary logs

Cloud Managed Services for Block Storage

Cloud Provider	Persistent Block Storage	Ephemeral Block Storage
Google Cloud	Persistent Disks	Local SSDs
AWS	Amazon EBS	Instance Store
Azure	Azure Managed Disks	Temporary Disks
Key Choice	Durability and reliability	Maximum performance but Temporary data

What is File Storage?

Scenario: Team collaborating on shared documents

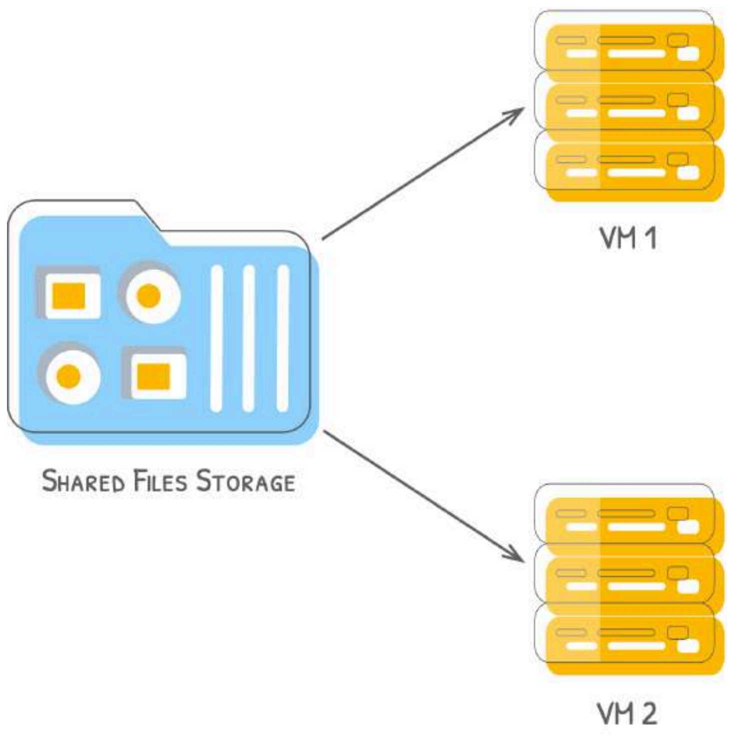
- **Need**: Accessible over network by multiple users

The Solution: File Storage

- Storage system organized in familiar folders and files

Key Protocols:

- **NFS**: Standard for Linux/Unix systems
- **SMB**: Standard for Windows environments



How does a File Storage Work?

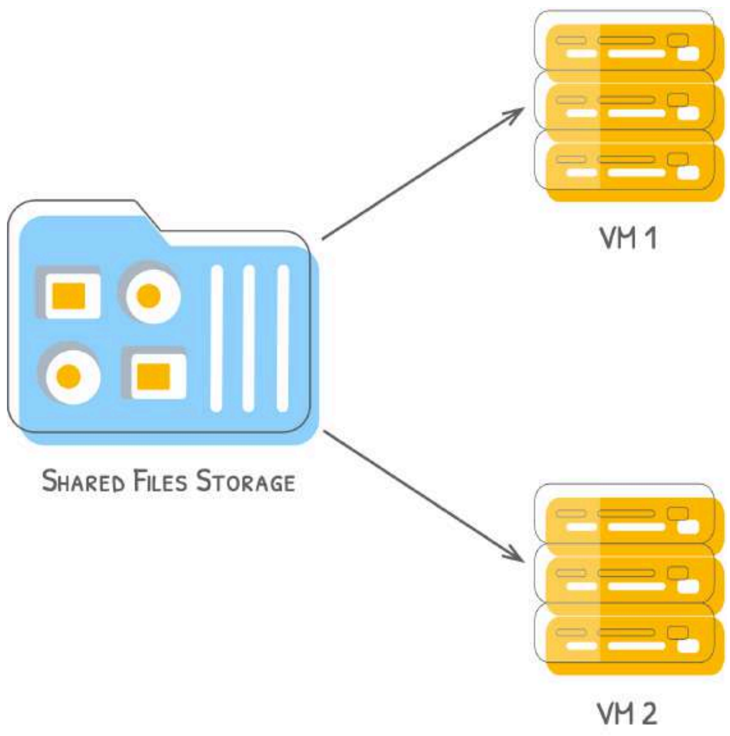
Mounted Volumes: File storage is mounted like a network drive

- **Folders & Files**: Organize data just like on your laptop

Benefits:

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

Cloud Services: Amazon EFS, Amazon FSx, Google Filestore, Azure Files



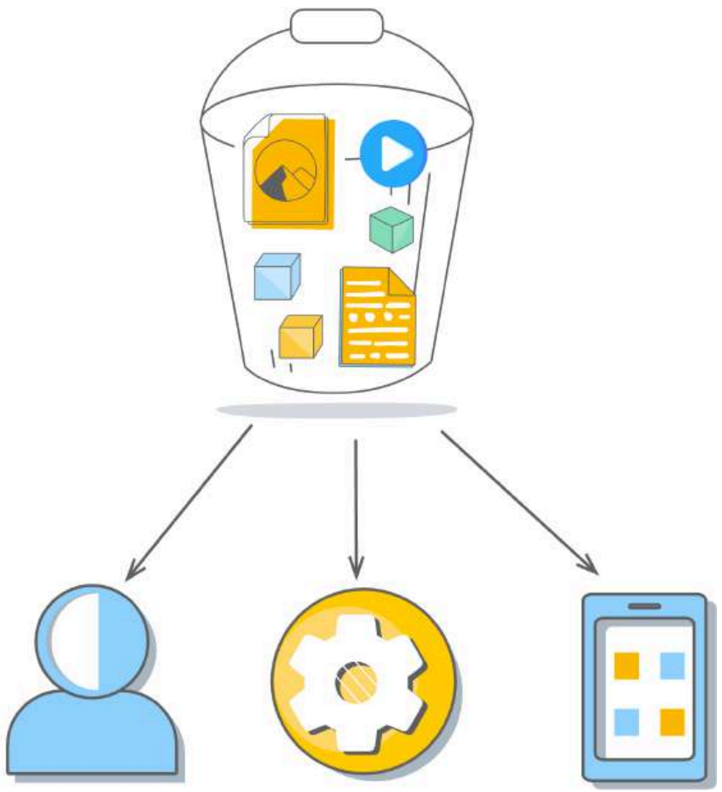
Why Object Storage in Cloud?

Scenario: Imagine millions of users uploading photos, videos, and documents through a website or mobile app

- **The Problem**: Traditional file systems struggle to scale globally

Object Storage: Designed for massive scale, durability, and web access

- **Access**: Retrieved via simple HTTP/REST APIs
- **Flat Structure**: No folders - everything stored in a **Bucket** with a unique key
- **Durable**: Automatically replicates data across zones or regions



Object Storage - Use Cases

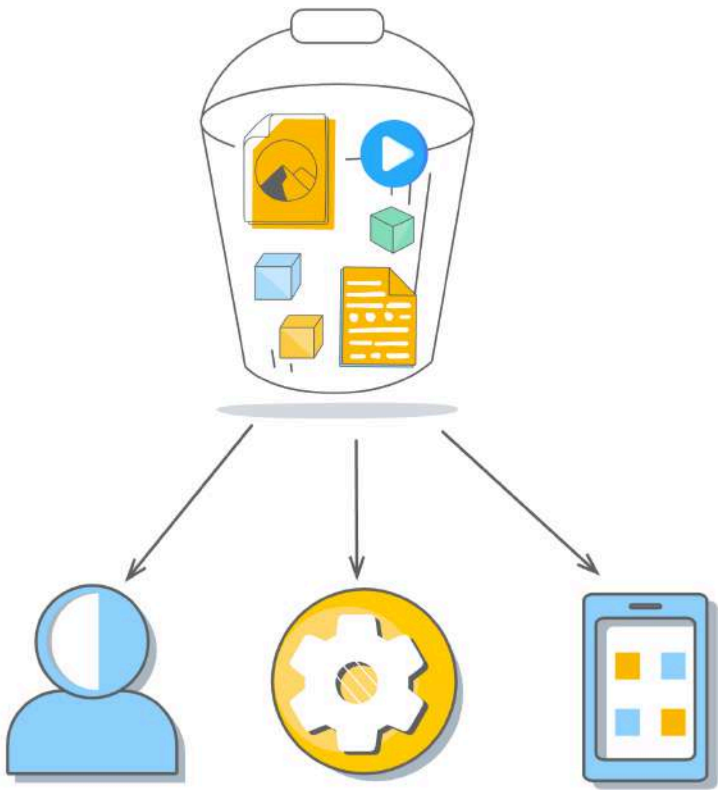
Object Storage: Most flexible storage type

Backup and Archiving: Reliable, long-term storage

Big Data and Analytics: Store large files for big data processing

Application Storage: Store logs, exports, reports

Serve Websites: Store and serve websites with images, videos, documents



Object Storage - Important Things to Remember

Highly Scalable Flat Structure: No folders

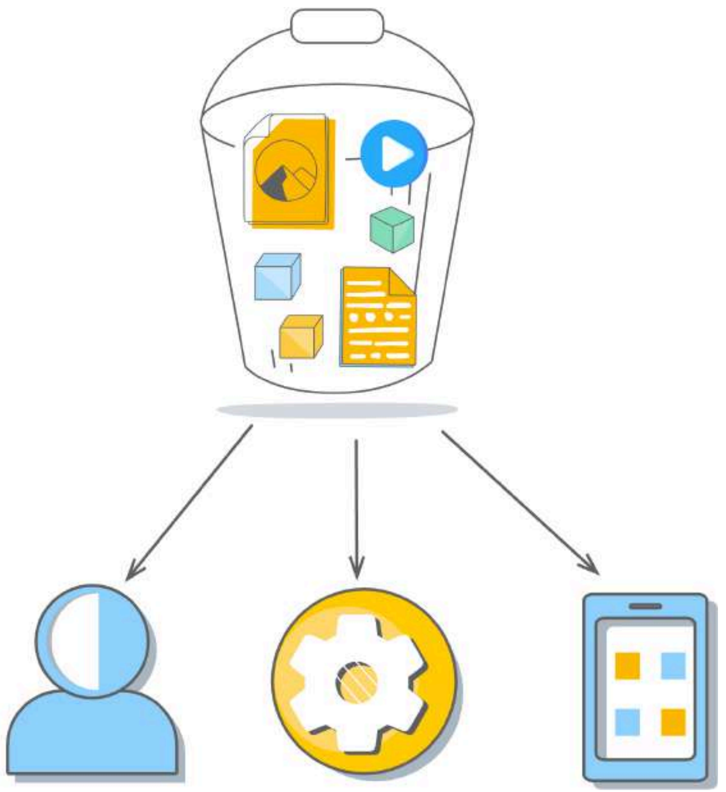
- **Key:** Everything stored in a bucket with a unique key
- **Scalable:** Can handle billions of files without performance loss

Durable: Data automatically replicated

- Across multiple zones (AZs) or regions

Access via HTTP APIs: Easy to integrate with applications, websites, and mobile apps

Managed Services: Amazon S3, Azure Blob Storage, Google Cloud Storage



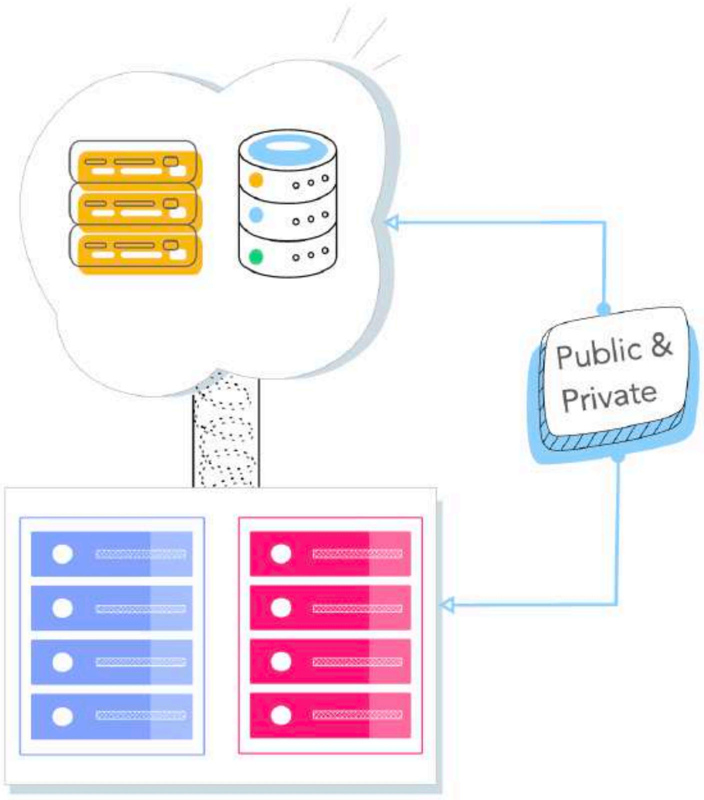
What is Hybrid Storage?

Scenario: Imagine storing large datasets needed from on-premises - some frequently accessed, some rarely touched

- Keeping all of it in the cloud may be slow
- Keeping it all on premises may limit scalability

Hybrid Storage: Bridge between on-premises storage and cloud storage

Use Cases: Gradual cloud migration, Archive and backup to cloud



Hybrid Storage - Advantages and Cloud Services

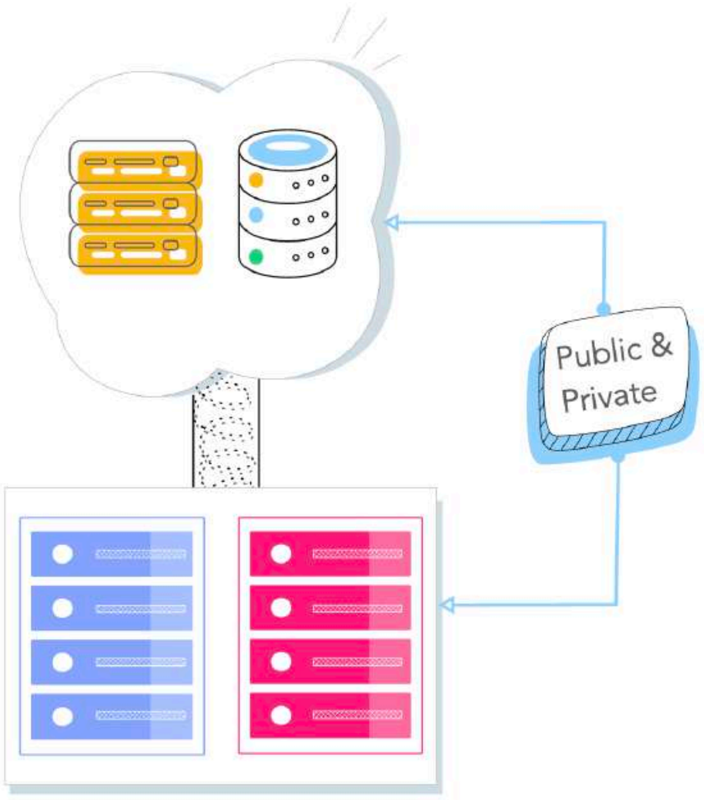
Scalability: Expand capacity without buying new hardware

Cost Efficiency: Store only hot data locally, cold data in the cloud

Disaster Recovery: Cloud backup improves disaster recovery

Cloud Services:

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



Storing Unstructured Data – Summary

Type	Best For	Access Method
Block Storage	OS, Databases, Low-latency disks	Attached as raw volume (Disk)
File Storage	Shared folders, Team collaboration	Mounted over network (NFS/SMB)
Object Storage	Web assets, media, backups, data lakes	Simple HTTP/REST APIs (No mounting)

Storing Unstructured Data – General Scenarios

Scenario	Solution
Storage Type: Install a custom OS for a unique application	Block Storage
Storage Type: Share config files across 10 Web Servers	File Storage
Storage Type: Store millions of user profile images for a mobile app	Object Storage
Storage Type: Booting a Linux VM instance	Block Storage
Persistent or Ephemeral Block Storage: Data that is recreated easily like temp caches	Ephemeral Block Storage