Interview QnA - Introduction to Storage in System Design

💬 Storage Fundamentals & Design Trade-offs

---

1. Why is storage a critical component in system design?
Storage is essential because it ensures data persistence across sessions and failures. All applications — from simple blogs to enterprise systems — need a reliable mechanism to store, retrieve, and update data. The choice of storage impacts scalability, performance, availability, and cost of the overall system.

---

2. How would you differentiate between structured and unstructured data?

- **Structured data** is highly organized and stored in a predefined schema (e.g., tables in SQL databases). Examples: customer records, transaction logs.
- **Unstructured data** lacks a fixed schema and is usually stored as raw files or blobs. Examples: images, videos, documents, social media posts.

---

3. What are the different types of storage systems and their use cases?

- **Databases (SQL/NoSQL):** For structured data, supporting queries and transactions
- **Object Storage:** For unstructured data like media, backups (e.g., Amazon S3)
- **File Storage:** Hierarchical storage, like shared drives (e.g., NFS, SMB)
- **Block Storage:** Raw storage volumes for high-performance apps (e.g., databases, VM disks)

⚙️ Properties of Storage Systems

---

4. What do durability, availability, and consistency mean in the context of storage?

- **Durability:** Data remains intact and recoverable after crashes or failures
- **Availability:** System continues to respond to requests even during failures
- **Consistency:** Clients always see the latest committed data after a write

---

5. What is atomicity, and where is it relevant?
Atomicity ensures that operations are **all-or-nothing** — they either complete fully or have no effect. It's crucial in **transactional systems** like databases where partial updates can lead to corruption (e.g., transferring money between accounts).

---

6. Can a system be both highly available and strongly consistent? Why or why not?
Not always — **CAP theorem** states that in the presence of a network partition, a distributed system must choose between consistency and availability. So, a system **can't guarantee both** during failures. Trade-offs are necessary based on business needs.

🔺 CAP Theorem Deep Dives

---

7. What is the CAP theorem? Why is it important in distributed system design?
The CAP theorem states that a distributed system can only guarantee **two out of three**:

- **Consistency**
- **Availability**
- **Partition Tolerance**

It's important because it guides architects in choosing trade-offs when designing distributed storage — especially under failure conditions.

---

8. Explain the difference between CP and AP systems with examples.

- **CP (Consistency + Partition Tolerance):** Prioritizes data accuracy over uptime. Example: **HBase**
- **AP (Availability + Partition Tolerance):** Prioritizes uptime, even if data is stale. Example: **DynamoDB**, **Couchbase**

---

9. Why is CA considered rare or impractical in distributed systems?
Because **network partitions are inevitable** in real-world distributed systems. Without partition tolerance, a system can't remain fault-tolerant. So CA systems only work in **centralized or tightly coupled systems** (e.g., single-node PostgreSQL).

---

10. How would you decide between consistency and availability when designing a real-world system?
It depends on the use case:

- For **financial systems**, **consistency** is non-negotiable
- For **social apps or media streaming**, **availability** is preferred — users can tolerate slight delays or stale views

---

🚀 Design Application Questions

---

11. You're building a photo-sharing app. How would you design storage for photos vs. user metadata?

- **Photos:** Store in **object storage** (e.g., S3), optimized for large unstructured files
- **Metadata (likes, comments, users):** Store in **NoSQL or relational database**, depending on query patterns and consistency needs

---

12. What kind of storage system would you choose for an analytics pipeline handling logs and metrics?
Use **append-optimized storage** with horizontal scalability. Examples:

- **Object storage** for raw logs (e.g., S3)
- **Columnar databases** or **time-series DBs** for metrics (e.g., Apache Druid, InfluxDB)
- **Distributed file systems** like HDFS for batch processing

---

13. How does object storage differ from file and block storage in terms of access patterns and scalability?

- **Object storage:** Scalable, flat structure, accessed via APIs, great for large datasets
- **File storage:** Hierarchical structure, good for shared directories
- **Block storage:** Raw, fast I/O, best for databases or OS-level operations