Interview Questions – System Reliability

🧠 Conceptual Questions

---

1. What is system reliability, and why is it important in system design?

Answer: System reliability refers to the ability of a system to consistently perform its intended function without failure over a specified period. In system design, reliability ensures:

- Minimal downtime
- Consistent user experience
- Protection of data and transactions
- Trust in the system's behavior under stress or failure

High reliability is essential in distributed systems, mission-critical applications (e.g., banking, healthcare), and cloud-native environments where failure can cascade across services.

---

2. Explain the difference between availability and durability with real-world examples.

Answer:

- **Availability** means the system is accessible and operational when needed.
    - *Example*: A website being online 24/7 with minimal downtime.
- **Durability** refers to the ability to retain and preserve data without loss.
    - *Example*: Data written to a cloud storage service like AWS S3 is not lost even if a node fails.

Analogy: Think of availability as whether the ATM is working, and durability as whether your money is still in your account.

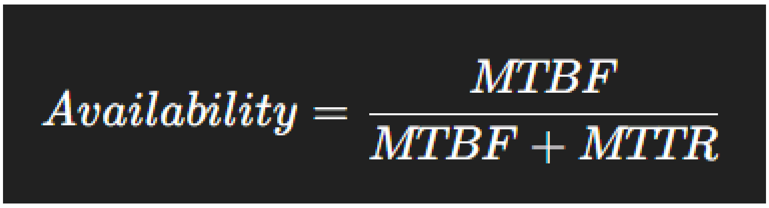
---

3. What are MTBF and MTTR? How do they relate to each other?

Answer:

- **MTBF (Mean Time Between Failures)**: Average time between two consecutive failures. It measures system reliability.
- **MTTR (Mean Time To Recovery)**: Average time to restore service after a failure. It measures resilience and repair efficiency.

Formula for Availability:



$$Availability = \frac{MTBF}{MTBF + MTTR}$$

The higher the MTBF and the lower the MTTR, the more reliable the system.

---

4. How do SLAs help define system reliability expectations?

Answer: SLAs (Service Level Agreements) are formal contracts between service providers and customers that define:

- Expected uptime/availability (e.g., 99.9% uptime)
- MTTR and support response times
- Penalties for failure to meet guarantees

They create clear reliability goals, help prioritize engineering efforts, and hold teams accountable for reliability metrics.

🔧 Practical / Scenario-Based

---

5. How would you design a system to ensure 99.99% availability?

Answer: To achieve 99.99% (aka "four nines") availability:

- Use **redundancy** (multiple instances/zones)
- Implement **load balancing** and **failover strategies**
- Design for **graceful degradation**
- Monitor with **alerting and auto-healing** systems
- Use **distributed storage** for high durability
- Continuously test with **chaos engineering**

Example: Use a multi-AZ setup in AWS with auto-scaling groups, health checks, and retry mechanisms.

---

6. Imagine one of your microservices goes down frequently. How would you identify and fix reliability issues?

Answer:

- **Step 1**: Check monitoring and alert logs for patterns (memory leaks, CPU spikes).
- **Step 2**: Analyze failure rate vs. expected MTBF.
- **Step 3**: Review recent code deployments and infrastructure changes.
- **Step 4**: Add circuit breakers, retries, or bulkheads to improve resilience.
- **Step 5**: Add observability (tracing, logging) to understand root causes.

Use SRE practices like Postmortems to prevent recurrence.

---

7. How would you improve reliability in a system with high user traffic and data volume?

Answer:

- **Scale horizontally** using stateless services
- **Partition/shard** databases to handle volume

- Use **message queues** to decouple services
- Implement **caching** to reduce load
- Optimize **MTTR** with autoscaling and self-healing
- Apply **rate limiting** and throttling

Example: Use Kafka for event ingestion, Redis for caching, and read replicas for query load.

---

8. Describe how redundancy and failover can be applied in cloud-native systems to improve reliability.

Answer:

- Deploy services in **multiple availability zones or regions**
- Use **health checks** with load balancers to reroute traffic
- Leverage **managed services** (e.g., RDS Multi-AZ)
- Automate failover using tools like AWS Route53 (geo DNS), Azure Traffic Manager
- Use **replication** and **backup recovery plans**

Redundancy prevents single points of failure, and failover ensures continuity.

⚖️ Behavioral / Trade-off Questions

---

9. Tell me about a time you had to choose between performance and reliability.

Answer Sample: "In a past project, we had a real-time analytics dashboard. We initially fetched live data directly from our primary DB to ensure up-to-the-second accuracy (high performance). However, it led to increased query load and affected reliability during traffic spikes. We chose to add a 15-second delay with a cache layer, slightly sacrificing real-time performance to greatly improve system stability and reduce MTTR during peak hours."

---

10. How would you ensure high reliability without over-engineering a system?

Answer:

- Start with **SLAs** to define clear reliability goals
- Apply the **Pareto Principle** (80/20 rule): fix 20% of issues causing 80% of failures
- Use **incremental improvements**: monitor → analyze → fix
- Avoid adding unnecessary complexity (e.g., too many layers or services)
- Prefer **managed cloud services** where possible

Focus on *simplicity, observability,* and *resilience* over theoretical perfection.