Mastering System Design

Section 9: Reliability, Availability & Disaster Recovery

Reliability- Section Agenda

1. Introduction to System Reliability
2. High Availability, Fault Tolerance & Failover
3. Backup & Recovery Strategies
4. Disaster Recovery in Practice
5. Summary and Recap: Building Reliable Distributed Systems

Introduction to System Reliability

Reliability, Availability & Disaster Recovery

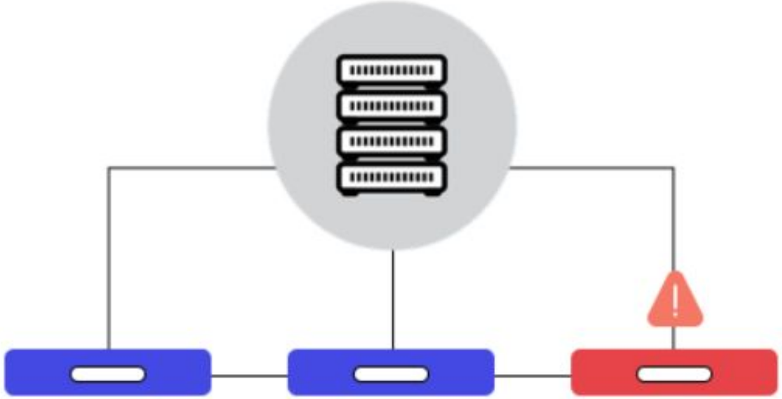
Why Reliability Matters

- Users expect systems to be always available, consistent, and trustworthy
- Downtime costs money, reputation, and user trust
- High reliability = low failure + fast recovery

“Systems fail. Reliability is how gracefully they handle it.”

What is System Reliability?

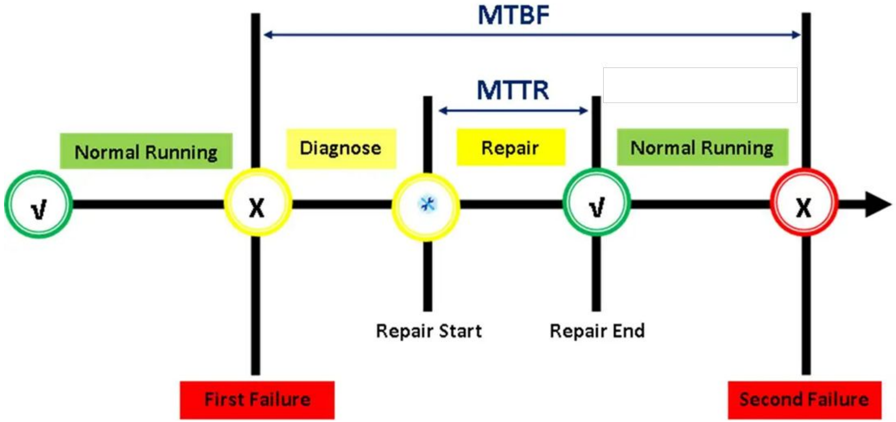
- Reliability is the ability of a system to operate continuously without failure
- It includes:
    - Correctness
    - Consistency
    - Fault tolerance
    - Uptime



Key Metrics - MTBF & MTTR

- MTBF: Mean Time Between Failures – Avg. time a system works before failing
- MTTR: Mean Time To Recovery – Avg. time to recover from failure

High MTBF + Low MTTR = High Reliability



What are SLAs (Service Level Agreements)?

- SLAs are contractual guarantees about system performance
- Common SLA metrics:
  - Availability % (e.g. 99.9%)
  - Response time
  - Error rate
- 99.9% uptime = ~8.76 hours of downtime/year

Availability vs. Durability

- Availability: System is accessible and responsive
- Durability: Data is safe and not lost

Impact of Reliability on System Design

- Design decisions that affect reliability:
  - Redundancy (multiple instances, failover)
  - Health checks and monitoring
  - Retry mechanisms and circuit breakers
  - Distributed design patterns (replication, quorum)

Reliability in Distributed Systems

- Challenges:
  - Network partitions
  - Node failures
  - Eventual consistency
- Solutions:
  - Use CAP-aware design
  - Ensure fault isolation
  - Implement replication & consensus algorithms (e.g. Paxos, Raft)

Reliability in Cloud-Native Systems

- Cloud infra is inherently unreliable
- Design for:
  - Transient failures
  - Auto-scaling and self-healing
  - Chaos engineering to test resilience

“Design for failure” is the cloud-native mindset

Interview Questions – System Reliability

- Conceptual Questions:
  - What is system reliability, and why is it important in system design?
  - Explain the difference between availability and durability with real-world examples.
  - What are MTBF and MTTR? How do they relate to each other?
  - How do SLAs help define system reliability expectations?
- Practical/Scenario-Based:
  - How would you design a system to ensure 99.99% availability?
  - Imagine one of your microservices goes down frequently. How would you identify and fix reliability issues?
  - How would you improve reliability in a system with high user traffic and data volume?
  - Describe how redundancy and failover can be applied in cloud-native systems to improve reliability.
- Behavioral/Trade-off Questions:
  - Tell me about a time you had to choose between performance and reliability.
  - How would you ensure high reliability without over-engineering a system?

Summary & Key Takeaways

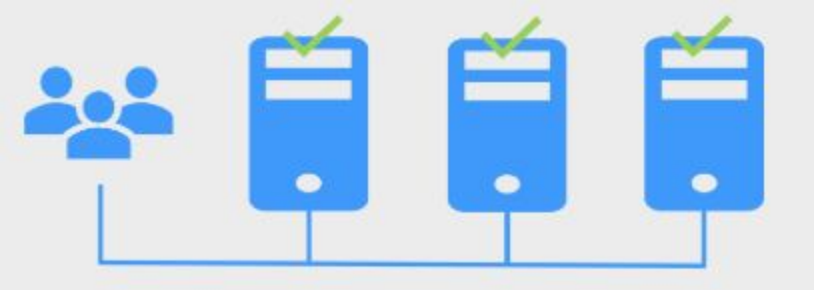
- Reliability is foundational to user trust
- Metrics like MTBF, MTTR, SLAs are vital to quantify it
- Design choices must anticipate and tolerate failure
- In distributed/cloud systems, reliability is an active design challenge
- What's next:
    - High Availability, Fault Tolerance & Failover

High Availability, Fault Tolerance & Failover

Reliability, Availability & Disaster Recovery

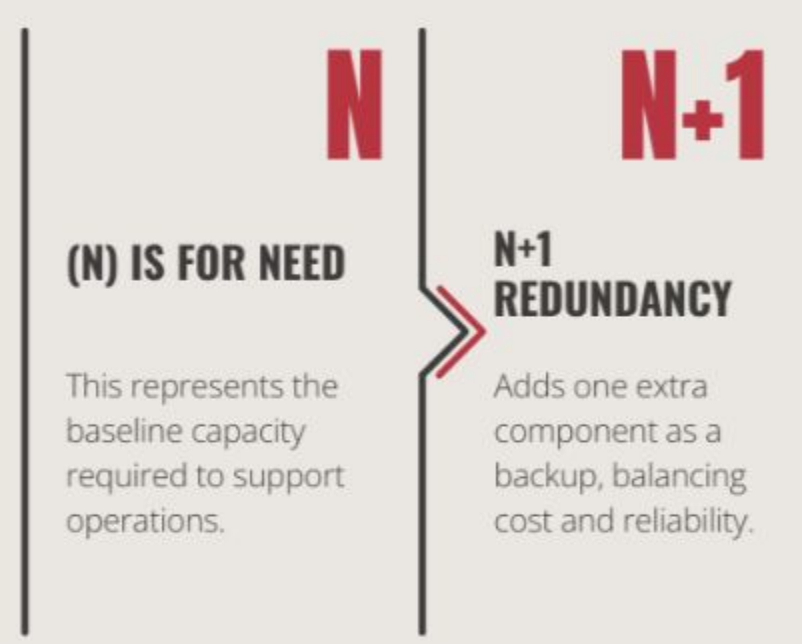
Redundancy and Redundancy Strategies

- Redundancy is crucial to ensuring system reliability and availability.
- Purpose: Prevent single points of failure.
- Types of redundancy:
  - Hardware (servers, storage)
  - Network (network routes, paths)
  - Services (microservices, databases)



N+1, Active-Active vs. Active-Passive

- N+1 Redundancy: One extra instance (e.g., 3 instances for 2 required)
    - Ensures availability in case of failure.
- Active-Active: Multiple nodes work together, each handling requests
    - High availability and load distribution.
- Active-Passive: One active node, others are standby, only activated on failure
    - Simpler but less efficient for load balancing.

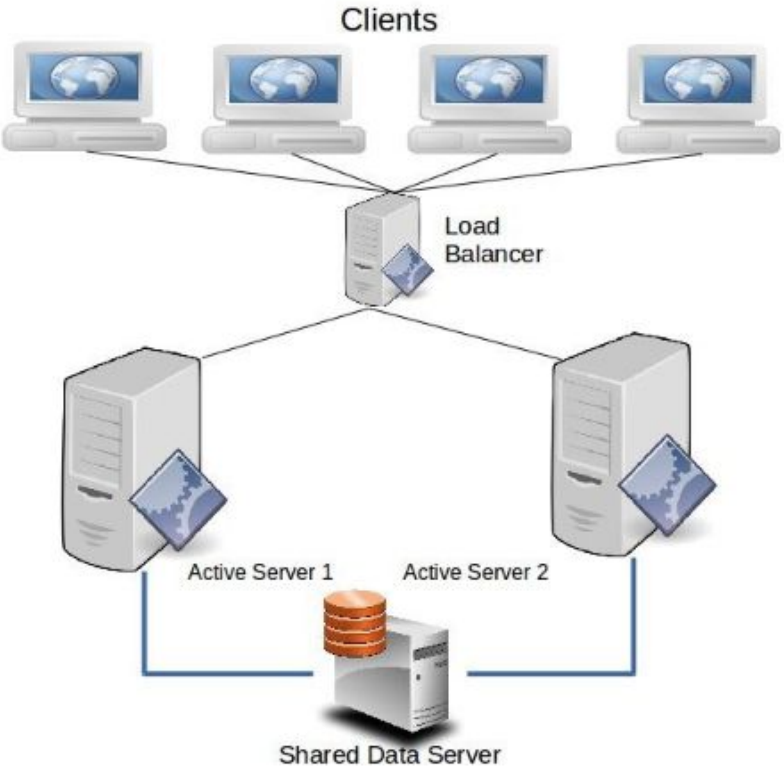


Graceful Degradation

- Graceful Degradation: System still operates at a reduced capacity during failures.
  - Example: During high traffic, disable non-essential features.
- Helps maintain user experience even when full service isn't possible.
- Critical for ensuring users still benefit from some functionality during outages.

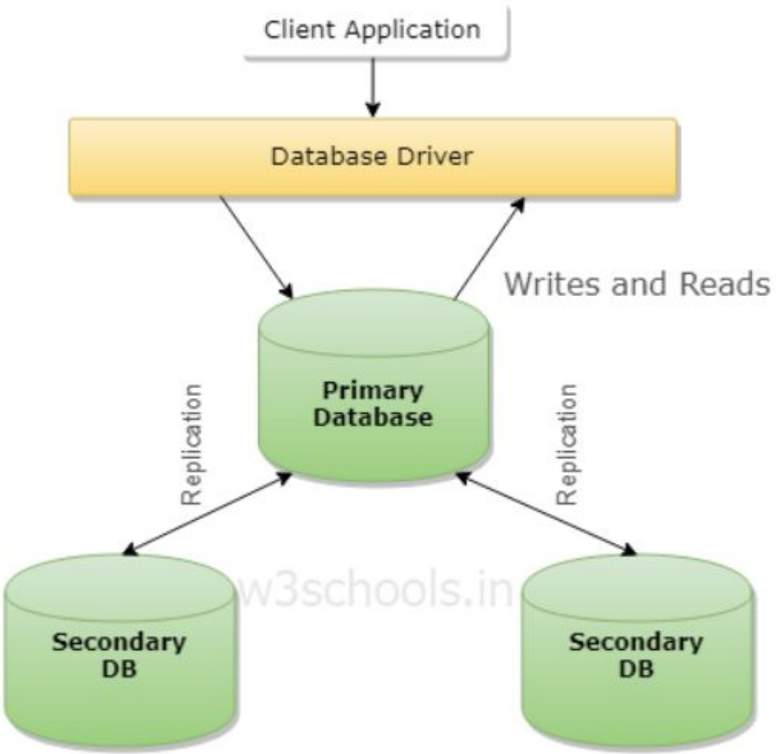
High Availability Patterns in Real-World Systems

- Load Balancers:
  - Distribute traffic evenly across healthy nodes.
  - Essential for active-active configurations.
- Replication: Copy data across multiple nodes or data centers for availability.
- Failover: Automatically switch to a backup node/service in case of failure.



Designing for Redundancy

- Redundant Components: Design with multiple instances of key components (servers, databases, etc.) to avoid failure.
- Geographical Redundancy: Use multiple data centers or cloud regions for disaster recovery.
- Automated Failover: Ensure failover happens automatically without manual intervention.



Health Monitoring & Self-Healing Systems

- Health Monitoring:
    - Track the status of system components (e.g., servers, services).
    - Alerts for system failures or performance degradation.
- Self-Healing Systems:
    - Automatically repair or replace failed components.
    - Commonly used in cloud environments for fault tolerance.

Interview Questions on High Availability, Fault Tolerance & Failover

- What is High Availability (HA) and why is it important in system design?
- Describe the difference between active-active and active-passive redundancy. When would you choose one over the other?
- What is fault tolerance, and how does it differ from high availability?
- What is graceful degradation, and how does it improve the user experience during a system failure?
- How do load balancers contribute to high availability in a distributed system?
- What is failover, and how does it help maintain availability during system failure?
- How can health monitoring and self-healing systems help improve system reliability and availability?

Summary & Key Takeaways

- High Availability ensures systems remain operational even in the event of failures through redundancy and failover.
- Fault Tolerance involves designing systems to continue functioning despite partial failures, using strategies like N+1 redundancy and graceful degradation.
- Failover systems provide automatic switching to backup systems to maintain service availability.
- Load Balancers play a crucial role in distributing traffic and ensuring HA across servers.
- Health Monitoring and Self-Healing Systems are critical for ensuring that systems can detect failures and recover autonomously.
- Designing systems with redundancy and resilience is essential to ensure availability and performance.
- What's next:
    - Backup & Recovery Strategies

Backup & Recovery Strategies

Reliability, Availability & Disaster Recovery

What is Backup & Recovery?

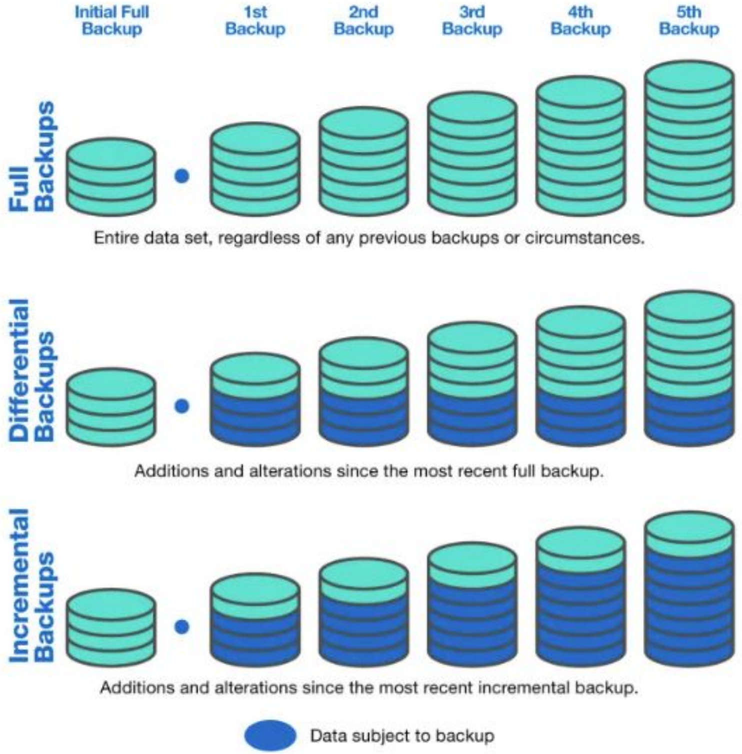
- Backup: Creating copies of data to protect against loss
- Recovery: Restoring data from backups after failure or corruption
- 🔁 Critical for disaster recovery, ransomware protection, hardware failures
- 🔐 Complements replication & redundancy

Why is Backup Important?

- Hardware or software failure
- Human error (accidental deletion)
- Cyber attacks (ransomware)
- Natural disasters
- Compliance & data retention

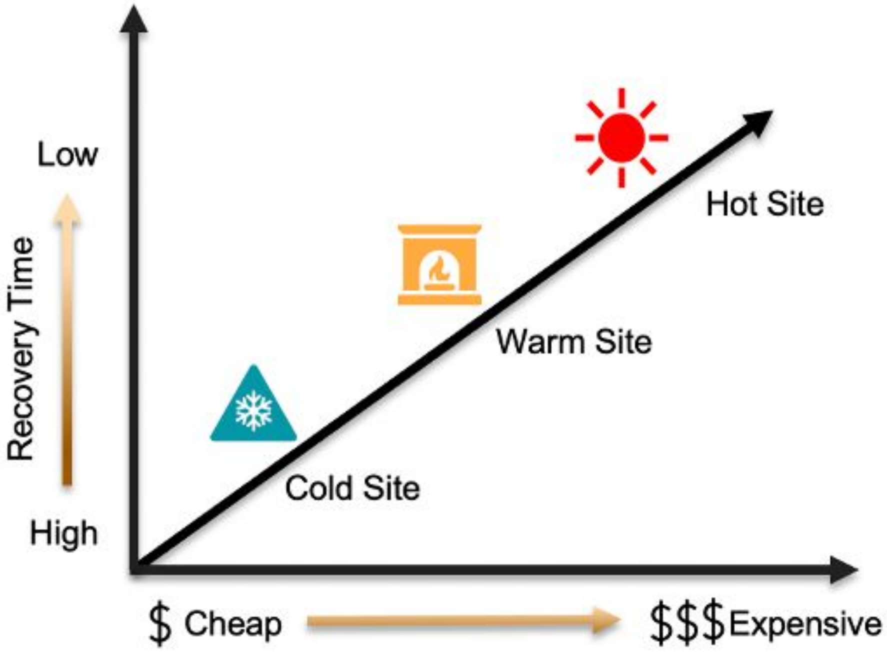
Types of Backup

- Full Backup
  - Copies all data
  - Simple recovery
  - High storage cost and time
- Incremental Backup
  - Backs up only changes since last backup (any type)
  - Small, fast, but needs all increments to restore
- Differential Backup
  - Backs up changes since last full backup
  - Bigger than incremental, faster restore



Recovery Types (Cold, Warm, Hot)

- Cold Recovery
  - Backups stored offline
  - High downtime, low cost
- Warm Recovery
  - Some resources pre-provisioned
  - Moderate downtime and cost
- Hot Recovery
  - Fully redundant, always-ready
  - Minimal downtime, highest cost



Understanding RTO & RPO

- RTO (Recovery Time Objective)
  - How quickly can the system recover?
- RPO (Recovery Point Objective)
  - How much data loss is acceptable?

🕓 Shorter RTO/RPO = Higher cost & complexity

Trade-offs in Backup Strategy

- 📉 Cost vs 🔁 Recovery speed
- 🧩 Complexity of managing backup frequency & retention
- ⚖️ Balance based on:
    - Business criticality
    - Compliance needs
    - Infrastructure maturity

Best Practices

- Automate backups and testing of restores
- Encrypt backups at rest and in transit
- Monitor backup success & failure
- Apply 3-2-1 Rule:
    - 3 copies
    - 2 different mediums
    - 1 offsite

Interview Questions – Backup & Recovery Strategies

- What is the difference between full, incremental, and differential backups? When would you use each?
- How do you define and balance RTO and RPO in a large-scale distributed system?
- Explain cold, warm, and hot recovery strategies with examples
- How would you implement a backup strategy for a microservices-based application hosted in the cloud?
- What trade-offs do you consider when designing a backup and recovery system for a high-availability service?
- How does cloud storage simplify or complicate backup and recovery strategies?
- What are some best practices for backup automation and testing in production systems?
- How would you handle backup for a database with terabytes of data and minimal allowed downtime?

Summary and Key Takeaways

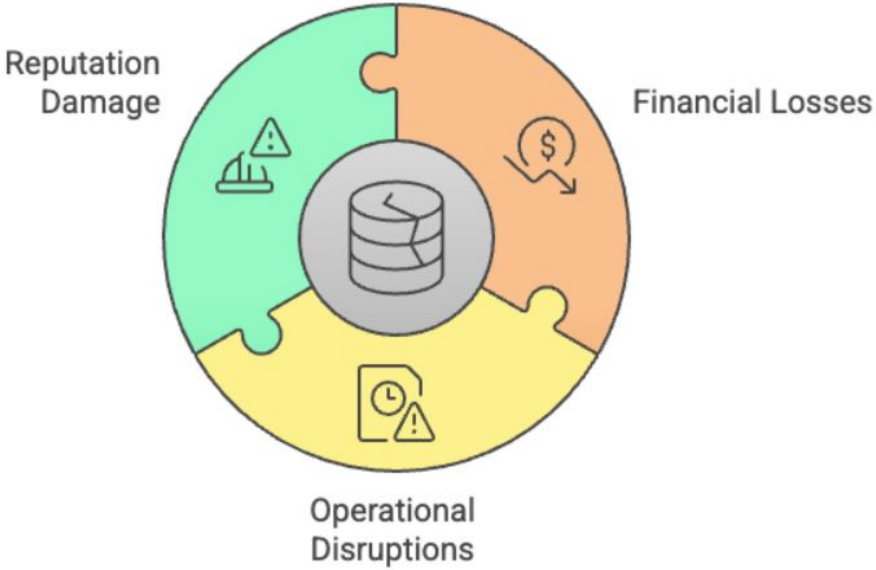
- Backups are vital for disaster recovery and resilience
- Choose the right backup type & recovery model
- Understand RTO/RPO goals
- Cloud solutions simplify but require governance
- Regular testing is key!
- What's next:
    - Disaster Recovery in Practice

Disaster Recovery in Practice

Reliability, Availability & Disaster Recovery

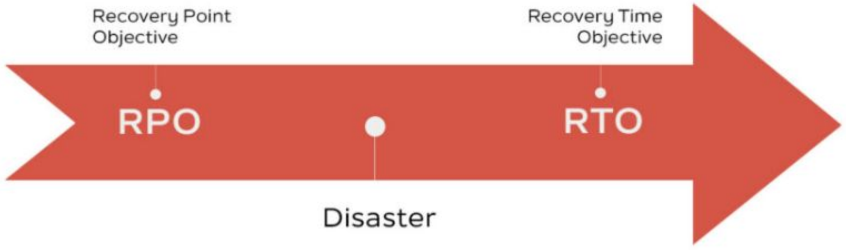
Why Disaster Recovery Matters

- Downtime costs $$$ — especially for mission-critical systems
- Protects against regional outages, data loss, and cyber attacks
- Complements backup strategies with full system-level resilience
- Required for compliance in regulated industries



DR for Mission-Critical Applications

- Systems must meet strict RTO & RPO targets
- Requires redundancy at multiple levels: compute, storage, network
- Automated failover + tested recovery plans
- Examples: banking, healthcare, e-commerce platforms



Failover + Backup = True Resilience

- Backups alone = data recovery
- Failover = service continuity
- Combine both:
    - Backups for corruption or deletion
    - Failover for infra or region outage
- Include both in DR plan for full coverage

Testing & Automation

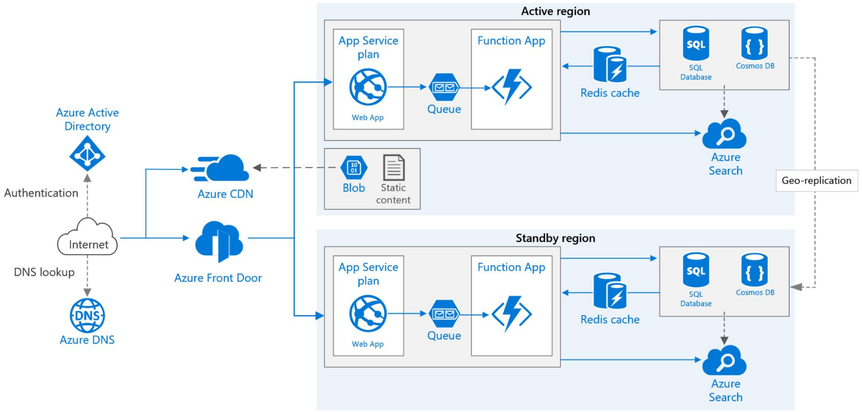
- DR plans must be tested regularly
- Automate:
  - Failover switching
  - Data validation after restore
  - Notifications & logging
- Run DR drills: simulate failures
- “If you haven’t tested it, you don’t have it”

Challenges in Geo-Distributed Systems

- Data consistency across regions
- Latency during sync and failover
- Regulatory constraints (data locality)
- Coordinating multi-region failovers

Geo-Redundancy & Quorum-Based Design

- Geo-redundancy: deploy across multiple physical locations
- Quorum-based design: Quorum is the minimum number of servers that must acknowledge a distributed operation to be successful before its marked a success.
- This ensures:
  - Data consistency
  - Safe failover (majority nodes available)



Interview Questions

- What’s the difference between failover and backup?
- How do you design DR for a high-traffic web app?
- What is RTO/RPO, and how do you optimize them?
- What are challenges with geo-distributed DR systems?
- Explain quorum-based design in distributed recovery

Summary and Key Takeaways

- DR is more than backups: it's about continuity
- Combine failover + backup for resilience
- Automate and test regularly
- Design for geo-redundancy with consistency mechanisms
- What's next:
  - Summary and Recap: Building Reliable Distributed Systems

Section Summary - Reliability, Availability & Disaster Recovery

- Introduction to System Reliability
  - Reliability concepts (MTBF, MTTR, SLAs, Availability vs. Durability)
  - The impact of reliability on system design and its role in cloud-native and distributed systems.
- High Availability, Fault Tolerance & Failover
  - Redundancy strategies like N+1, active-active, and active-passive
  - Graceful degradation, HA patterns, and designing for redundancy with health monitoring and self-healing systems.
- Backup & Recovery Strategies
  - Different types of backups (full, incremental, differential)
  - Understanding RTO/RPO and trade-offs in backup strategies
  - Cloud-based backup strategies for cost-effective, scalable recovery.
- Disaster Recovery in Practice
  - Designing DR for mission-critical apps and learning from a real-world case study
  - Combining failover and backup for complete resilience
  - Testing and automating recovery plans, and challenges with geo-distributed systems and quorum-based designs.
- What’s next:
  - Security in System Design