





Shalini Goyal
@goyalshalini



The Ultimate System Design Guide

**Master the 10 Core Areas for
Interviews & Architecture**



Scalable Data Storage

| | |
|-----------------------------|--|
| Partitioning | Vertical vs. Horizontal Sharding - Vertical sharding splits by functionality, while horizontal sharding distributes data across nodes for scalability. |
| SQL vs. NoSQL | When to Pick Which and Why - SQL ensures structured consistency; NoSQL handles high-velocity, flexible schema data. Choose based on workload needs. |
| Indexing | Indexing: Primary, Secondary, and Covering Indexes - Speed up queries using primary (unique identifiers), secondary (for searches), and covering indexes (reducing disk reads). |
| Consistency Models | Strong, Eventual, and Causal Consistency - Strong ensures immediate updates; eventual allows latency for availability; causal tracks dependencies for ordering. |
| Replication | Primary-Secondary, Leader-Follower Configurations - Distribute read operations efficiently while ensuring data consistency across multiple nodes. |
| Data Modeling | Schema Design for Relational and NoSQL Databases - Design schemas to optimize queries, relations, and data integrity for structured and unstructured data. |
| Storage Optimization | Compression and Data Deduplication Techniques - Reduce storage costs and speed up data retrieval by eliminating redundancies and compressing large datasets. |



Shalini Goyal
@goyalshalini



Caching

| | |
|------------------------------------|---|
| Cache Invalidation | Consistency Between Cache and Source Data - Ensure stale data is refreshed by implementing proper cache invalidation policies. |
| Client-Side vs. Server-Side | Client-Side vs. Server-Side: Choosing the Right Caching Layer - Client-side caching reduces request overhead, while server-side caching optimizes backend performance. |
| Application-Specific Cache | Application-Specific Cache: Query and Result-Level Caching - Improve efficiency by caching frequent queries and precomputed results at the application level. |
| Strategies | Write-Through, Write-Back, Write-Around - Write-through synchronizes cache and storage; write-back delays writes for speed; write-around bypasses cache on writes. |
| Distributed Cache | Redis, Memcached, and Beyond - Use in-memory storage systems to accelerate access to frequently used data across distributed applications. |
| Cache Metrics | Hit Ratios, Latency Monitoring - Track cache effectiveness by monitoring hit ratios, eviction rates, and request latencies. |
| Eviction Policies | LRU, LFU, Random Eviction - Remove old cache entries efficiently based on usage frequency or randomized strategies. |



Shalini Goyal
@goyalshalini



Load Balancing

| | |
|------------------------------|---|
| Sticky Sessions | When and Why to Use Them - Maintain session consistency by ensuring requests from the same user are directed to the same server. |
| Reverse Proxies | Tools Like Nginx and HAProxy - Improve security and performance by managing traffic before it reaches backend servers. |
| Horizontal Scaling | Expanding Services Dynamically - Add more servers to distribute load and prevent performance bottlenecks. |
| Failover Mechanisms | Handling Downtime Gracefully - Ensure high availability by redirecting traffic to backup servers when failures occur. |
| Global Load Balancing | Multi-Region Traffic Distribution - Route users to the closest data center to reduce latency and improve reliability. |
| Health Monitoring | Ensuring Backend Servers Are Always Up - Use heartbeat checks and health probes to detect failures early. |
| Techniques | Round-Robin, Consistent Hashing, Least Connections - Optimize load balancing using different traffic distribution methods. |



Shalini Goyal
@goyalshalini



Asynchronous Processing

| | |
|----------------------------------|---|
| Event-Driven Architecture | Benefits and Implementation Patterns - Decouple services by using events to trigger actions asynchronously. |
| Pub/Sub Models | Decoupling Producers and Consumers - Enable real-time communication by allowing multiple consumers to process events. |
| Dead Letter Queues | Handling Undeliverable Messages - Prevent message loss by capturing failed events for later processing. |
| Scalability | Partitioning and Scaling Message Brokers - Improve throughput by distributing messages across partitions and multiple brokers. |
| Message Brokers | Kafka, RabbitMQ, and SQS - Manage message queues effectively for event-driven processing. |
| Task Queues | Managing Retries and Timeouts - Implement job queues for background processing and ensure failed jobs are retried. |
| Stream Processing | Real-Time Data Handling - Process and analyze data continuously with tools like Apache Flink and Spark Streaming. |



Shalini Goyal
@goyalshalini



Database Read and Write Scaling

| | |
|---|---|
| Write Partitioning | Challenges and Solutions - Distribute write loads to multiple nodes while managing consistency and performance. |
| Read Replicas | Scaling Read Operations Effectively - Improve read scalability by distributing queries across replicas. |
| Quorum-Based Writes | Achieving Strong Consistency - Ensure fault tolerance by requiring a majority of nodes to acknowledge writes. |
| Multi-Region Replication | Ensuring Global Availability - Distribute databases across regions to reduce latency and increase fault tolerance. |
| Consistency Trade-Offs | Balancing Latency and Correctness - Choose between consistency and availability depending on application needs. |
| Optimistic vs. Pessimistic Locking | Conflict Resolution Strategies - Prevent race conditions with optimistic (retry conflicts) or pessimistic (lock resources) approaches. |
| Leader-Follower Patterns | Handling Writes in Distributed Systems - Maintain a single authoritative source for writes while allowing read scaling. |



Shalini Goyal
@goyalshalini



Distributed Systems Concepts

| | |
|---------------------------------|--|
| Consensus Algorithms | Paxos, Raft for Distributed Coordination - Ensure agreement among nodes for reliable distributed operations. |
| Distributed Transactions | Two-Phase and Three-Phase Commit - Maintain data integrity across distributed databases by coordinating commits. |
| Leader Election | Zookeeper and Kubernetes Implementations - Dynamically choose a leader node to manage distributed tasks. |
| Partition Tolerance | Handling Network Splits Effectively - Design systems to remain available even when network failures occur. |
| Data Replication | Techniques for High Availability - Copy data across nodes to ensure redundancy and fault tolerance. |
| Conflict Resolution | CRDTs, Vector Clocks, LWW - Resolve data conflicts in distributed databases using different reconciliation methods. |
| Gossip Protocols | Decentralized Data Sharing - Exchange state information across nodes efficiently without central coordination. |



Shalini Goyal
@goyalshalini



Reliability and Failover

| | |
|-----------------------------|--|
| Retries | Handling Transient Failures Gracefully - Implement retry logic with exponential backoff to avoid overloading systems. |
| Auto-Healing Systems | Self-Recovery Mechanisms - Detect and automatically recover from failures with minimal downtime. |
| Redundancy | Active-Active vs. Active-Passive Setups - Distribute workloads across multiple live or standby backup servers. |
| Replication Lag | Monitoring and Minimizing Delays - Optimize data replication to ensure minimal delay between database copies. |
| Disaster Recovery | Planning for Worst-Case Scenarios - Implement data backups, failovers, and contingency plans to handle failures. |
| Circuit Breakers | Preventing Cascading System Failures - Avoid system crashes by blocking failing services temporarily. |
| Health Checks | Monitoring Application and System Health - Ensure continuous operation with automated status checks. |



Shalini Goyal
@goyalshalini



Content Delivery Networks (CDNs)

| | |
|---------------------------------|---|
| Dynamic Content Delivery | Dynamic Content Delivery: Handling API Responses via Edge Nodes - Optimize API calls by caching responses closer to users. |
| Performance Monitoring | Tools to Evaluate CDN Effectiveness - Analyze load times and cache efficiency for performance tuning. |
| Cache Purging | Keeping Content Fresh at Edge Locations - Automate cache invalidation for frequently changing content. |
| Geo-Load Balancing | Direct Traffic to the Nearest Server - Improve performance by routing users to the closest edge location. |
| Static Content Delivery | Reduce Latency with CDNs - Serve images, scripts, and videos quickly through edge caching. |
| Edge Caching | Reduce Latency with CDNs - Serve images, scripts, and videos quickly through edge caching. |
| Origin Shielding | Protecting Backend Servers - Reduce load on origin servers by using a caching layer as a buffer. |



Shalini Goyal
@goyalshalini



API Design and Rate Management

| | |
|---------------------------------|---|
| Throttling | Preventing Abuse and Overload - Control API usage to prevent excessive requests from overloading servers. |
| Authentication | OAuth, API Keys, and JWT - Secure APIs using different authentication mechanisms. |
| Pagination and Filtering | Efficiently Fetch Data - Optimize API responses by limiting results and allowing custom filtering. |
| REST vs. GraphQL | Choosing the Right Paradigm - REST offers structured endpoints; GraphQL provides flexible queries. |
| Error Handling | Designing Graceful Error Responses - Return informative error messages for better debugging. |
| API Versioning | Smooth Transitions for Evolving APIs - Manage API updates without breaking existing clients. |
| Rate Limiting | Algorithms Like Token Bucket, Leaky Bucket - Control traffic flow and prevent system abuse. |



Shalini Goyal
@goyalshalini



Search Systems

| | |
|--------------------------------------|---|
| Indexing | Techniques for Fast Data Retrieval - Improve search speed by structuring data into efficient index trees, such as B-Trees or inverted indexes. |
| Full-Text Search Engines | Elasticsearch, Solr - Enable scalable and fast text-based searches with distributed search engines optimized for large datasets. |
| Search Sharding | Distribute Queries Efficiently - Split search indexes across multiple nodes to handle high query loads and improve response times. |
| Ranking and Relevance | Improve Search Quality - Use algorithms like TF-IDF, BM25, and learning-to-rank models to return the most relevant results. |
| Synonyms and Stemming | Enhance Search Results - Expand search queries to include variations of words, improving accuracy and user experience. |
| Autocomplete Systems | Real-Time Suggestion Mechanisms - Provide instant query suggestions using predictive indexing and prefix-based lookups. |
| Monitoring Search Performance | Latency and Relevance Tracking - Analyze query times, result accuracy, and indexing efficiency to optimize search systems. |



Shalini Goyal
@goyalshalini





Shalini Goyal

@goyalshalini

Follow for more!