Jean Malaquias • Following
🚀 Tech Entrepreneur | AI & Growth Strategist | Founder | Startup Mentor | Cloud & Produ...
5d •

Top 20 System Design Concepts You Should Know

1 - Load Balancing: Distributes traffic across multiple servers for reliability and availability.

2 - Caching: Stores frequently accessed data in memory for faster access.

3 - Database Sharding: Splits databases to handle large-scale data growth.

4 - Replication: Copies data across replicas for availability and fault tolerance.

5 - CAP Theorem: Trade-off between consistency, availability, and partition tolerance.

6 - Consistent Hashing: Distributes load evenly in dynamic server environments.

7 - Message Queues: Decouples services using asynchronous event-driven architecture.

8 - Rate Limiting: Controls request frequency to prevent system overload.

9 - API Gateway: Centralized entry point for routing API requests.

10 - Microservices: Breaks systems into independent, loosely coupled services.

11 - Service Discovery: Locates services dynamically in distributed systems.

12 - CDN: Delivers content from edge servers for speed.

13 - Database Indexing: Speeds up queries by indexing important fields.

14 - Data Partitioning: Divides data across nodes for scalability and performance.

15 - Eventual Consistency: Guarantees consistency over time in distributed databases

16 - WebSockets: Enables bi-directional communication for live updates.

17 - Scalability: Increases capacity by upgrading or adding machines.

18 - Fault Tolerance: Ensures system availability during hardware/software failures.

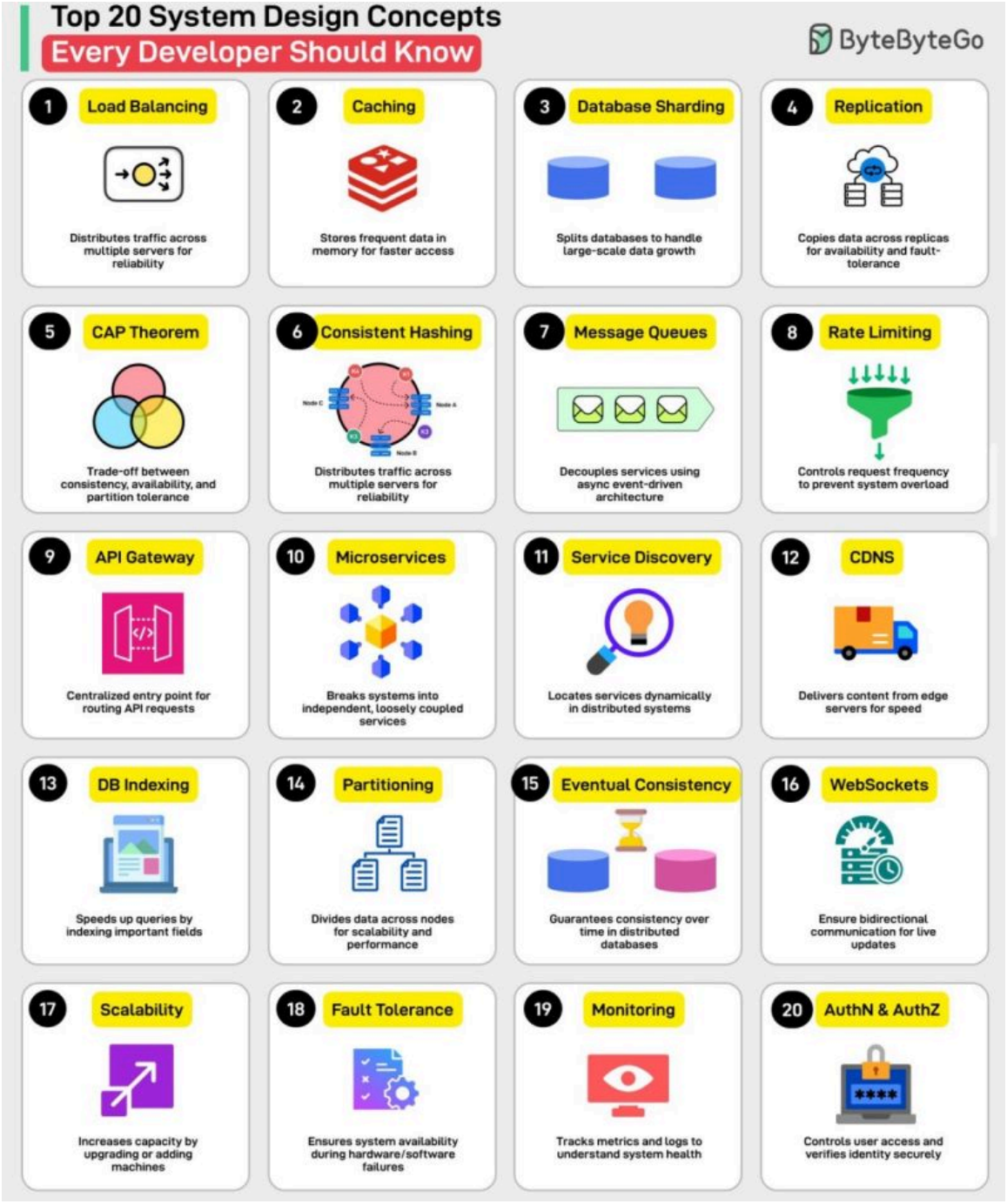
19 - Monitoring: Tracks metrics and logs to understand system health.

20 - Authentication & Authorization: Controls user access and verifies identity securely.

Over to you: Which other System Design concept will you add to the list?

--

Source: Alex Xu


Top 20 System Design Concepts
Every Developer Should Know
ByteByteGo
1 Load Balancing
Distributes traffic across multiple servers for reliability
2 Caching
Stores frequent data in memory for faster access
3 Database Sharding
Splits databases to handle large-scale data growth
4 Replication
Copies data across replicas for availability and fault-tolerance
5 CAP Theorem
Trade-off between consistency, availability, and partition tolerance
6 Consistent Hashing
Distributes traffic across multiple servers for reliability
7 Message Queues
Decouples services using async event-driven architecture
8 Rate Limiting
Controls request frequency to prevent system overload
9 API Gateway
Centralized entry point for routing API requests
10 Microservices
Breaks systems into independent, loosely coupled services
11 Service Discovery
Locates services dynamically in distributed systems
12 CDNS
Delivers content from edge servers for speed
13 DB Indexing
Speeds up queries by indexing important fields
14 Partitioning
Divides data across nodes for scalability and performance
15 Eventual Consistency
Guarantees consistency over time in distributed databases
16 WebSockets
Ensure bidirectional communication for live updates
17 Scalability
Increases capacity by upgrading or adding machines
18 Fault Tolerance
Ensures system availability during hardware/software failures
19 Monitoring
Tracks metrics and logs to understand system health
20 AuthN & AuthZ
Controls user access and verifies identity securely

17