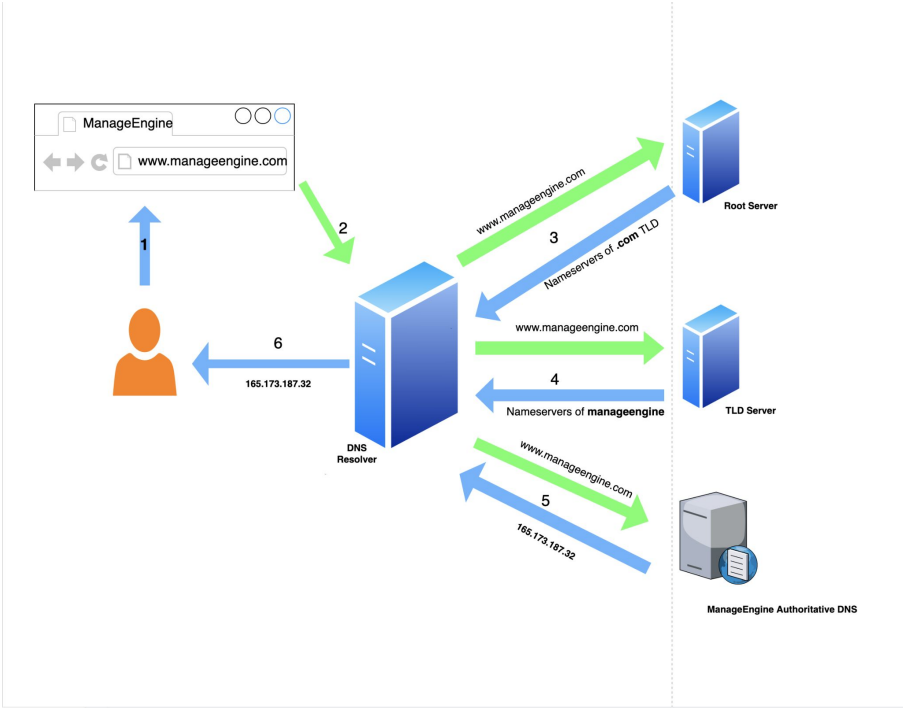
System Design Components

05/2023

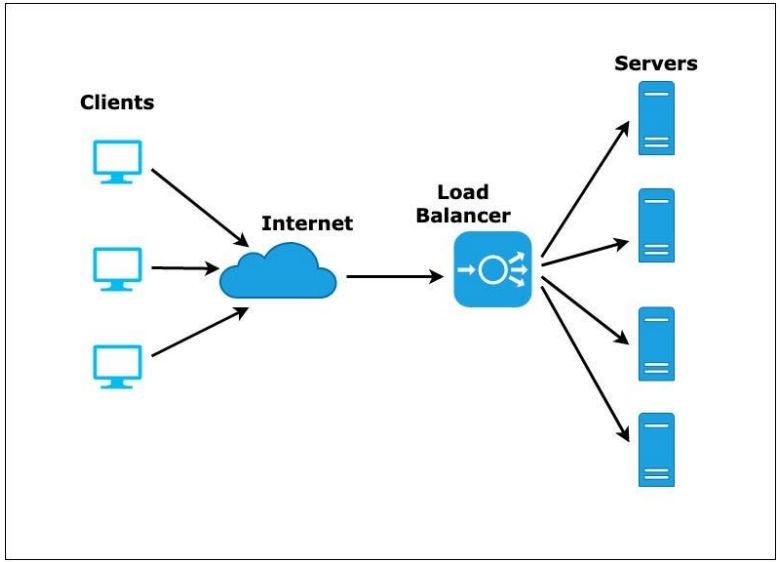
1. Domain Name System (DNS)

The DNS is a hierarchical and distributed system used to translate human-readable domain names, such as www.example.com, into IP addresses, like 192.168.1.1, that computers use to identify each other on the internet or a private network. The primary purpose of DNS is to make it easier for users to access websites and other network resources by using meaningful and easy-to-remember domain names instead of having to remember numerical IP addresses.



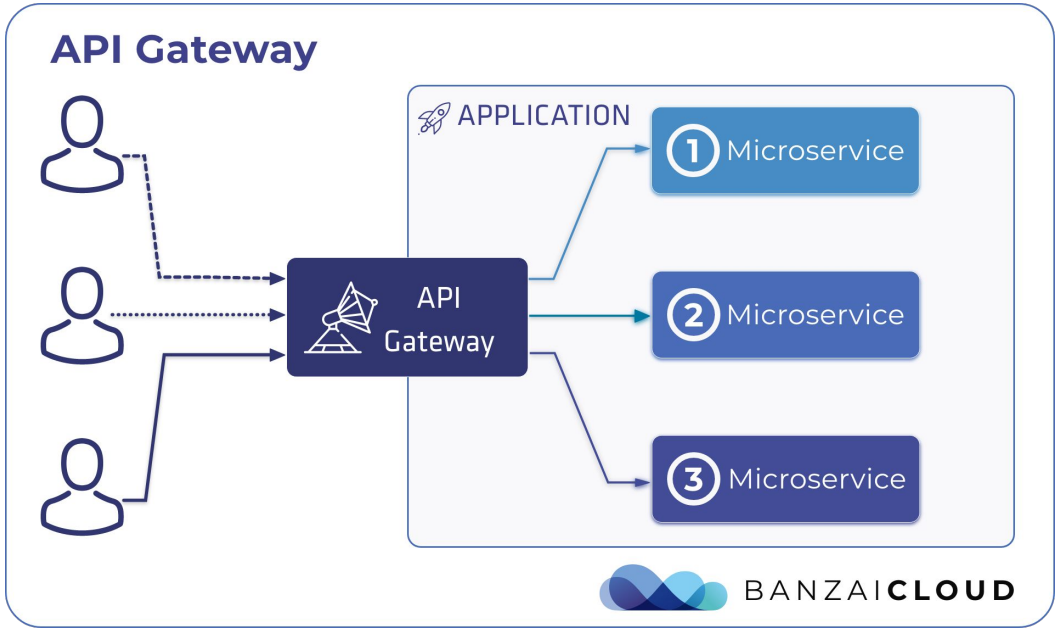
2. Load Balancer

A load balancer is a network device that distributes incoming network traffic across multiple backend servers or services to improve the performance and availability of the system. The load balancer typically sits between the client and the server and uses various algorithms to distribute incoming requests among the available servers in a manner that maximizes performance and ensures that no single server is overwhelmed. This can improve the overall reliability and responsiveness of the system, as it allows for a more even distribution of workload and enables the system to handle a higher volume of requests.



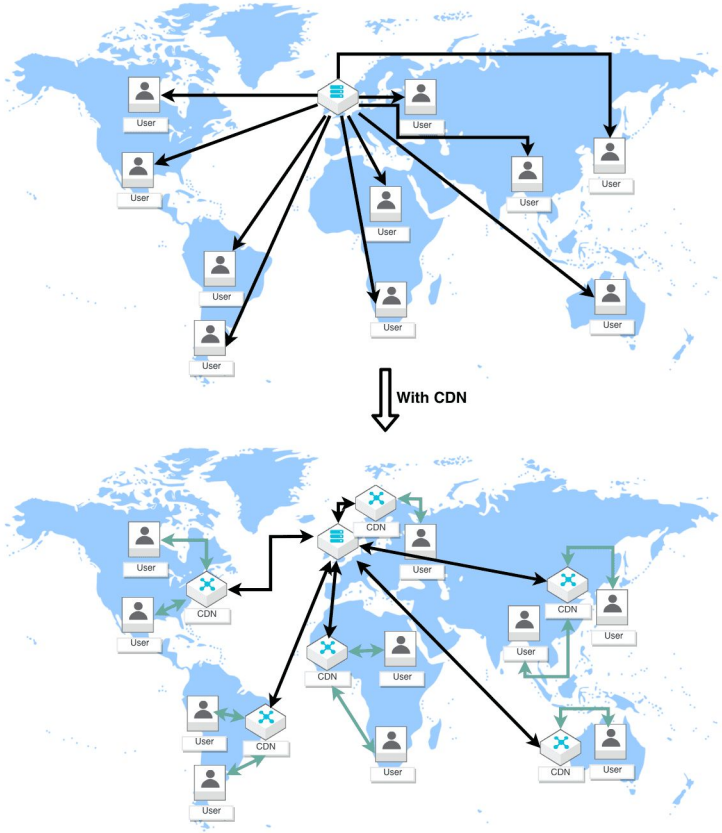
3. API Gateway

An API Gateway (AG) is a server that acts as a single point of entry for a set of microservices. AG receives client requests, forwards them to the appropriate microservice, and then returns the server's response to the client. AG is responsible for tasks such as routing, authentication, and rate limiting.



4. Content Delivery Network (CDN)

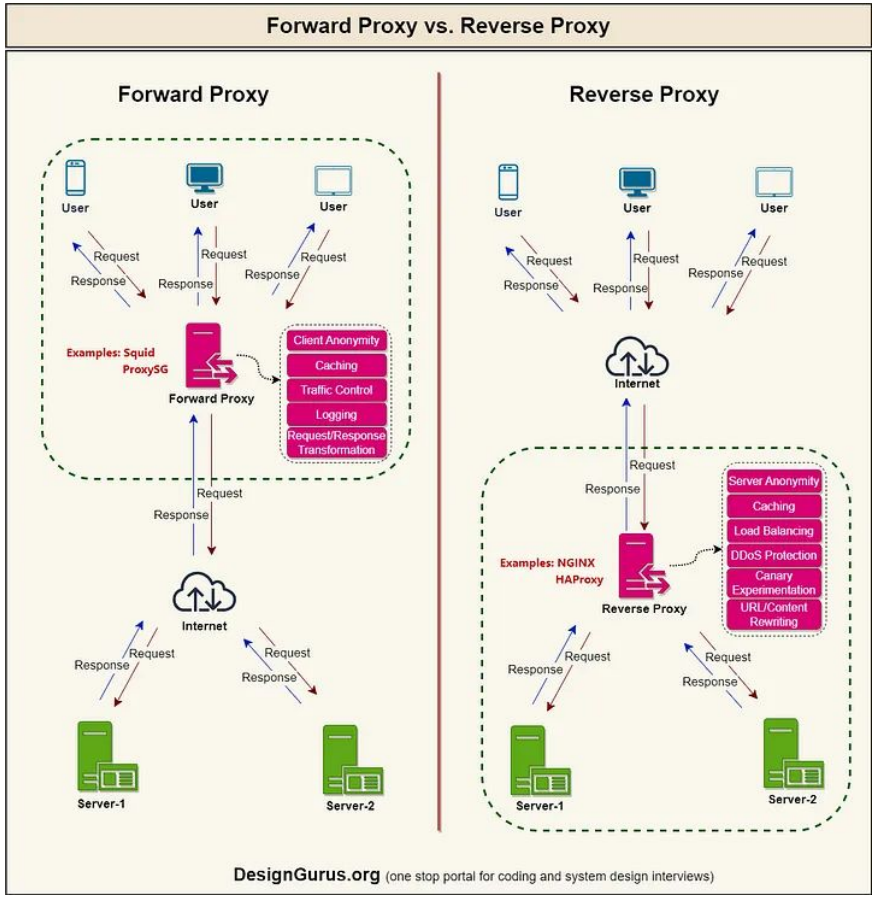
A Content Delivery Network (CDN) is a distributed network of servers that are deployed in multiple locations around the world. These servers are designed to deliver web content, such as images, videos, and other static files, to users based on their geographical location. The main purpose of a CDN is to improve the performance and availability of web content by caching it on servers that are closer to the users who are requesting it.



5. Forward Proxy vs Backward Proxy

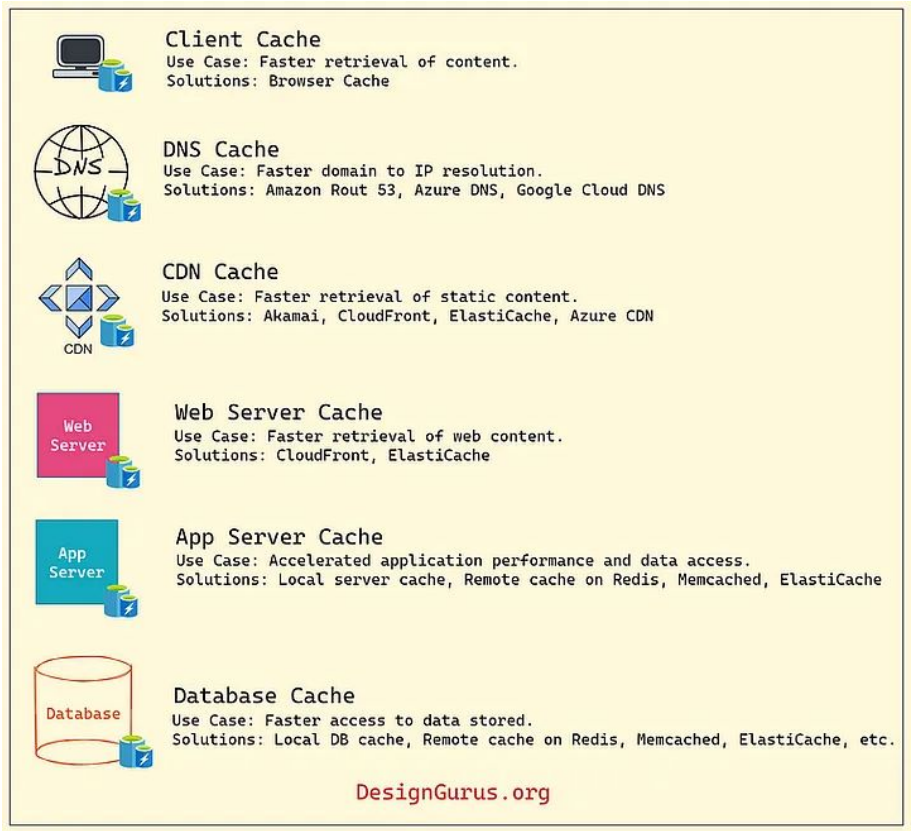
A **forward proxy**, also known as a “proxy server,” or simply “proxy,” is a server that sits in front of one or more client machines and acts as an intermediary between the clients and the internet. When a client machine makes a request to a resource on the internet, the request is first sent to the forward proxy. The forward proxy then forwards the request to the internet on behalf of the client machine and returns the response to the client machine.

A **reverse proxy** is a server that sits in front of one or more web servers and acts as an intermediary between the web servers and the Internet. When a client makes a request to a resource on the internet, the request is first sent to the reverse proxy. The reverse proxy then forwards the request to one of the web servers, which returns the response to the reverse proxy. The reverse proxy then returns the response to the client.



6. Caching

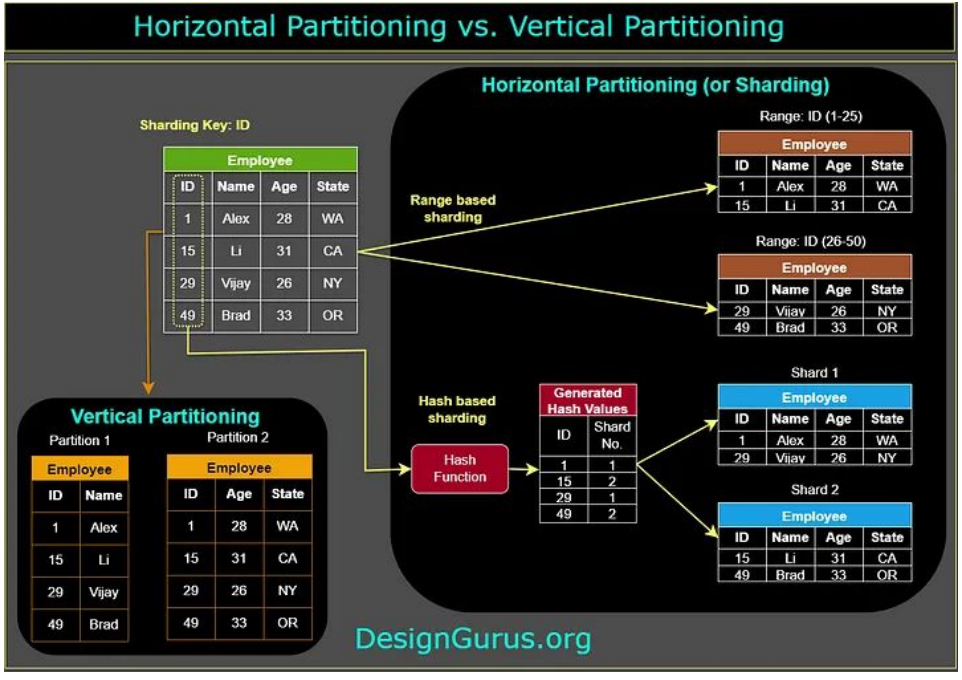
The cache is a high-speed storage layer that sits between the application and the original source of the data, such as a database, a file system, or a remote web service. When data is requested by the application, it is first checked in the cache. If the data is found in the cache, it is returned to the application. If the data is not found in the cache, it is retrieved from its original source, stored in the cache for future use, and returned to the application. In a distributed system, caching can be done at multiple places for example, Client, DNS, CDN, Load Balancer, API Gateway, Server, Database, etc.



7. Data Partitioning

In a database, **horizontal partitioning**, also known as **sharding**, involves dividing the rows of a table into smaller tables and storing them on different servers or database instances. This is done to distribute the load of a database across multiple servers and to improve performance.

On the other hand, **vertical partitioning**, involves dividing the columns of a table into separate tables. This is done to reduce the number of columns in a table and to improve the performance of queries that only access a small number of columns.



8. Sharding

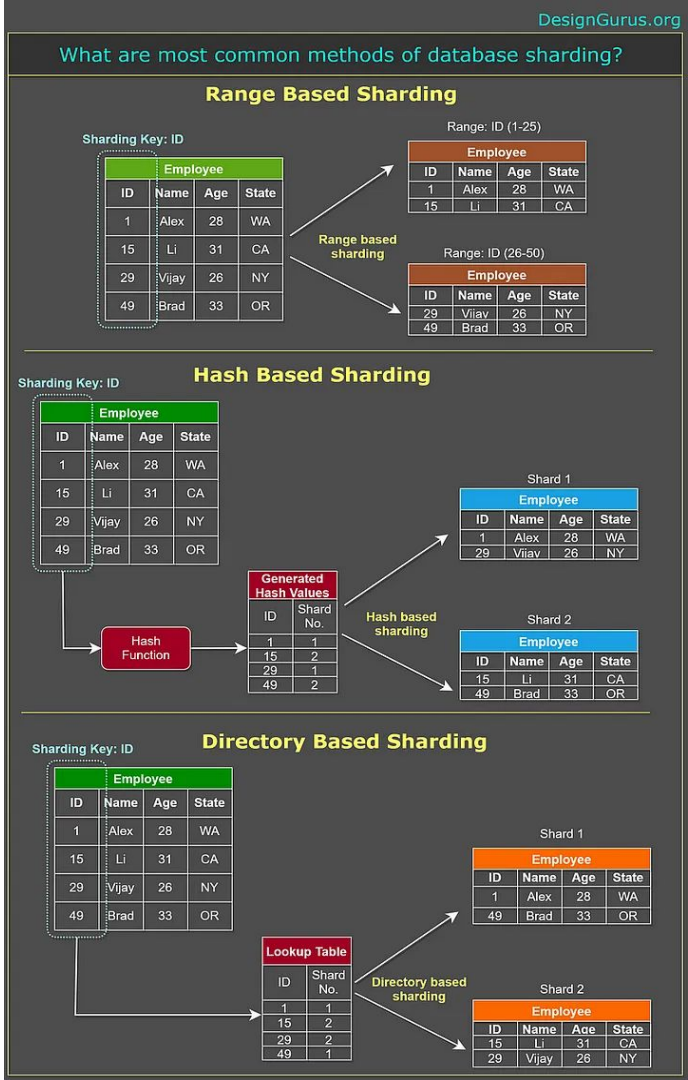
It is a technique used to scale a database by horizontally partitioning the data across multiple servers, or shards. The goal of sharding is to distribute the data and workload across multiple servers, so that each server can handle a smaller portion of the overall data and workload. This can help improve the performance and scalability of the database, as each server can process queries and updates more efficiently when it is working with a smaller amount of data.

Range-based sharding: In this approach, the data is partitioned based on a key value, such as a user ID or a timestamp, and the data is distributed across the shards based on the range of the key value. For example, all user IDs in the range of 1–1000 might be stored on one shard, while user IDs in the range of 1001–2000 might be stored on another shard.

Hash-based sharding: In this approach, a hash function is used to distribute the data across the shards based on the key value. For example, all data with a user ID of 123 might be stored on one shard, while data with a user ID of 456 might be stored on another shard.

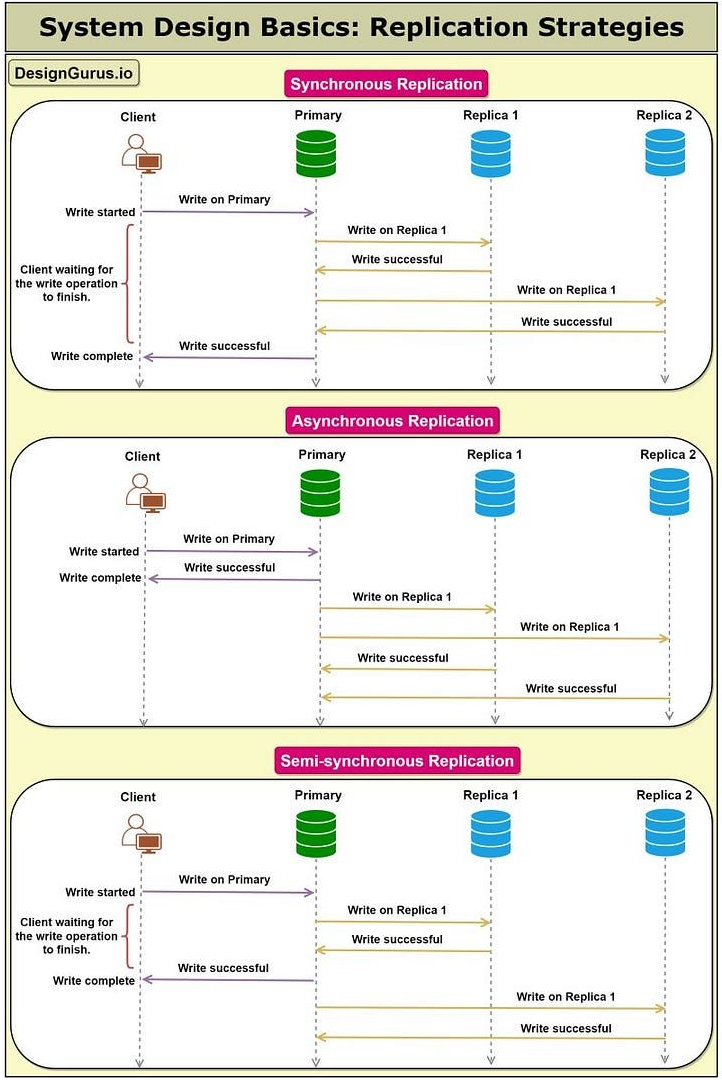
Directory-based sharding: In this approach, a central directory is used to map the key values to the specific shard where the data is stored. The directory can be used to determine which shard a piece of data belongs to, and the data can be retrieved from the appropriate shard.

Custom sharding: In some cases, it may be necessary to implement a custom sharding strategy that is specific to the needs and requirements of the database and the applications that are using it. This can involve a combination of different sharding methods or a completely unique approach.



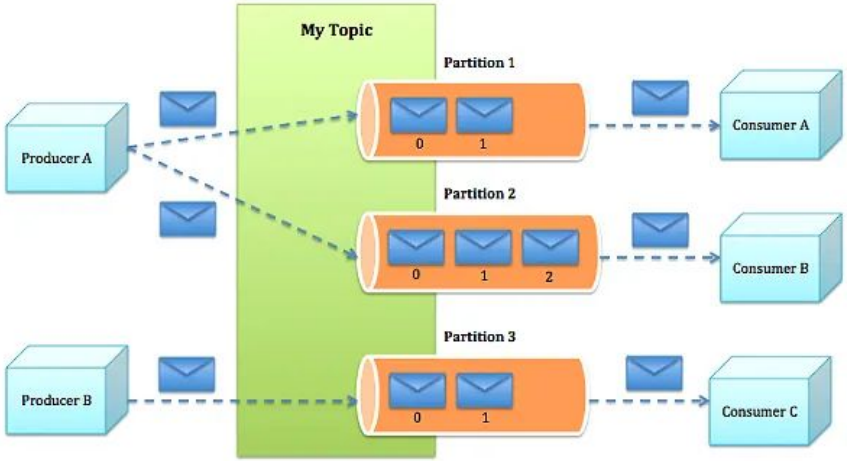
9. Database Replication

Database replication is the process of copying and synchronizing data from one database to one or more additional databases. This is commonly used in distributed systems where multiple copies of the same data are required to ensure data availability, fault tolerance, and scalability. Here are the top three typical database replication strategies:



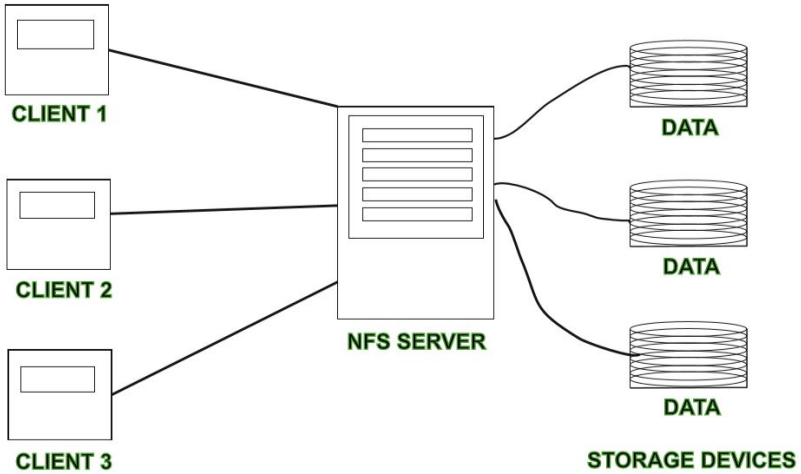
10. Distributed Messaging System

Distributed messaging systems enable the exchange of messages between multiple, potentially geographically-dispersed applications, services, or components in a reliable, scalable, and fault-tolerant manner. They facilitate communication by decoupling the sender and receiver components, allowing them to evolve and operate independently. Distributed messaging systems are particularly useful in large-scale or complex systems, such as those found in microservices architectures or distributed computing environments. Examples of such systems are Apache Kafka and RabbitMQ.



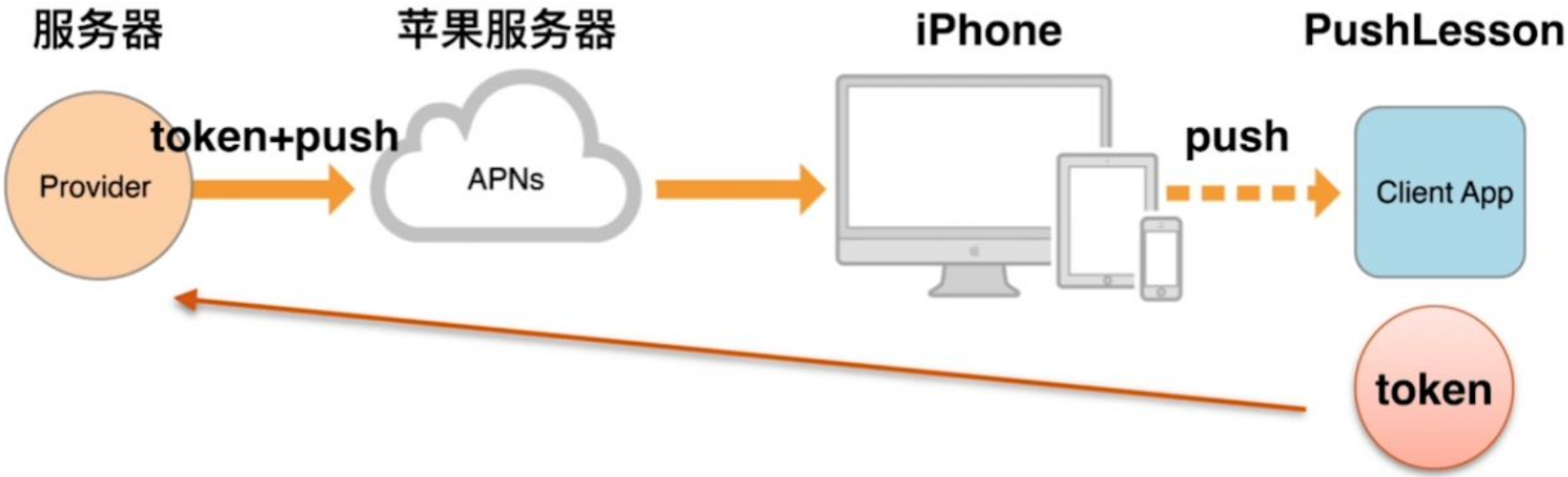
11. Distributed File Systems

Distributed file systems are storage solutions designed to manage and provide access to files and directories across multiple servers, nodes, or machines, often distributed over a network. They enable users and applications to access and manipulate files as if they were stored on a local file system, even though the actual files might be physically stored on multiple remote servers. Distributed file systems are often used in large-scale or distributed computing environments to provide fault tolerance, high availability, and improved performance.



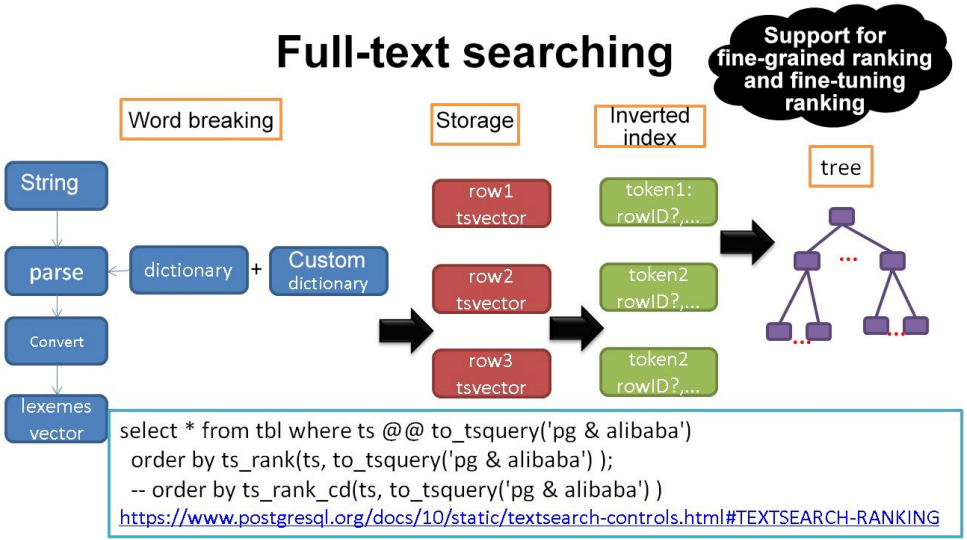
12. Notification System

These are used to send notifications or alerts to users, such as emails, push notifications, or text messages.



13. Full-text Search

Full-text search enables users to search for specific words or phrases within an app or website. When a user queries, the app or website returns the most relevant results. To do this quickly and efficiently, full-text search relies on an inverted index, which is a data structure that maps words or phrases to the documents in which they appear. An example of such systems is Elastic Search.



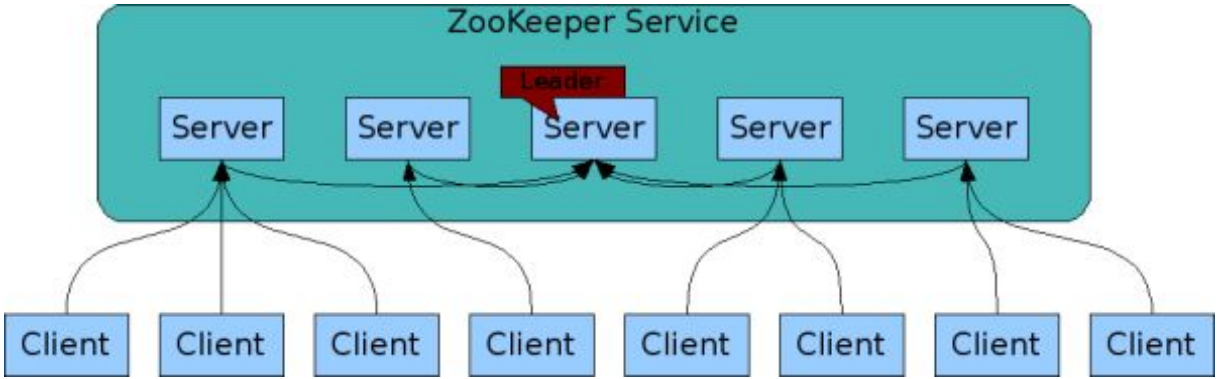
14. Data Warehouse

A data warehouse is a large, centralized repository of structured and historical data used for reporting, analysis, and decision-making purposes within an organization. It collects, stores, and manages data from various sources, such as transactional databases, external data sources, and application systems, and organizes it in a way that facilitates efficient querying and reporting.

Data warehouses are designed to support the efficient retrieval and analysis of large volumes of data, enabling organizations to gain insights and make data-driven decisions. They typically store data in a highly structured format, often using a schema design, such as the star or snowflake schema, to optimize query performance.

15. Distributed Coordination Services

Distributed coordination services are systems designed to manage and coordinate the activities of distributed applications, services, or nodes in a reliable, efficient, and fault-tolerant manner. They help maintain consistency, handle distributed synchronization, and manage the configuration and state of various components in a distributed environment. Distributed coordination services are particularly useful in large-scale or complex systems, such as those found in microservices architectures, distributed computing environments, or clustered databases. Examples of such service are Apache ZooKeeper, etcd, Consul.



Reference

- https://levelup.gitconnected.com/system-design-master-template-how-to-answer-any-system-design-interview-question-ee5dc332acd5#0768