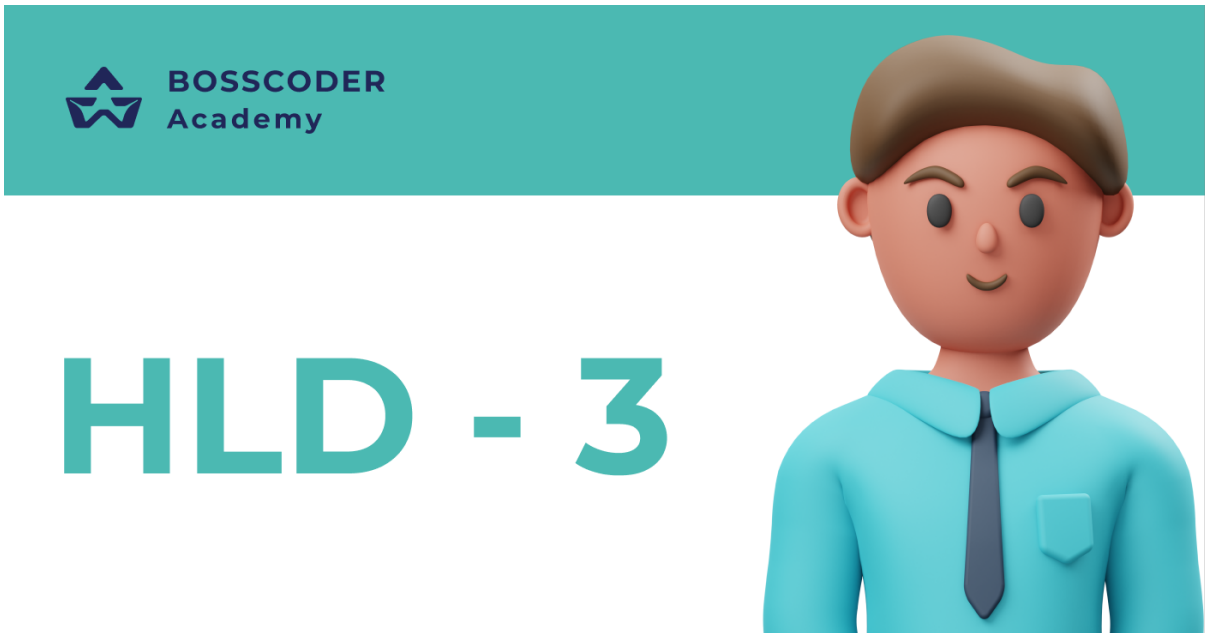
BOSSCODER ACADEMY

Request A Callback



HLD - 3

Abhishek Singh
5th August, 2023

High-Level Design

High-Level Design (HLD) is a system architecture blueprint that defines the overall structure and components of a complex system. It focuses on key elements, interactions, and high-level decisions. HLD provides an abstract representation of the system's functionality, modules, and their relationships, without diving into implementation details.

In the context of high-level design, a proxy acts as an intermediary between clients and servers, providing additional functionalities, such as caching, load balancing, and security. Proxies help improve system performance, scalability, and security by handling client requests on behalf of servers.

Reverse Proxy

A reverse proxy is a type of proxy that manages incoming client requests and forwards them to one or more backend servers. It operates on behalf of the server and is typically placed between the clients and backend servers. Reverse proxies are commonly used to improve security, handle SSL/TLS termination, and distribute traffic among multiple servers.

Load Balancer

A load balancer is a critical component in high-level design, responsible for distributing incoming network traffic across multiple servers or nodes. It ensures efficient utilization of resources, prevents server overloading, and provides fault tolerance by redirecting traffic away from unhealthy servers.

Types of Load Balancers:

1. **Random Load Balancer:** Selects a server randomly from the available pool to handle each incoming request, without considering server load or capacity.
2. **Round Robin Load Balancer:** Cycles through the available servers sequentially, distributing incoming requests equally among them.
3. **Weighted Round Robin Load Balancer:** Assigns weights to servers

4. **Health Check Load Balancer:** Monitors the health of servers and routes requests only to healthy servers, effectively removing unhealthy servers from the pool.
5. **IP-Based Load Balancer:** Distributes traffic based on the source IP address of the client, ensuring a particular client is consistently directed to the same server.
6. **Path-Based Load Balancer:** Routes requests to different servers based on the URL path requested by the client, allowing for specific routing rules for different endpoints.

Hashing

Hashing is a technique used to map data to fixed-size values (hash codes) representing the original data. In high-level design, hashing is often used for data distribution in distributed systems, ensuring even data distribution across multiple nodes.

Consistent Hashing

Consistent hashing is a variation of hashing designed to minimize data movement and rebalancing when nodes are added or removed from a distributed system. It ensures that the distribution of data across nodes remains stable even when the number of nodes changes, making it well-suited for scalable systems.

Database

The high-level design includes the selection and design of the appropriate database system to store and manage the application's data effectively.

NoSQL Databases:

Data Model:

- Relational databases use a tabular data model with predefined schemas, where data is organized into tables with rows and columns. Each row represents a record, and each column represents an attribute of the record.
- NoSQL databases, including NoDocument databases, use flexible data models. They can store unstructured or semi-structured data, making them ideal for handling dynamic or rapidly evolving data.

Scalability:

- Relational databases are vertically scalable, meaning they can be upgraded with more powerful hardware resources, but they have limits to scalability due to the rigid table structures.
- NoSQL databases, particularly NoDocument databases, are horizontally scalable, meaning they can handle increased data volume by adding more nodes to the cluster, making them suitable for large-scale distributed systems.

Schema Flexibility:

- Relational databases require a fixed schema, meaning the structure of tables and relationships between them must be defined beforehand and remain consistent.
- NoSQL databases, especially NoDocument databases, offer schema flexibility, allowing developers to store varying data types without

Query Language:

- Relational databases use SQL (Structured Query Language) to interact with the data, which is standardized and widely used in the industry.
- NoSQL databases may use different query languages or APIs, depending on the specific database type and vendor.

Consistency and ACID Properties:

- Relational databases are known for maintaining ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring strong consistency and transactional support.
- NoSQL databases, particularly those designed for high availability and scalability, may trade some of the ACID properties for better performance and availability, offering eventual consistency instead.

Use Cases:

- Relational databases are well-suited for applications that require complex data relationships and transactions, such as banking systems and financial applications.
- NoSQL databases, including NoDocument databases, are ideal for applications that handle large volumes of unstructured or semi-structured data, like social media platforms, IoT applications, and content management systems.

High-level design decisions regarding database selection involve considering factors like data structure, scalability, performance, and consistency needs to meet the application's specific requirements effectively.