Client Server Architecture

What is client server architecture?

Client-server architecture is a type of distributed system that consists of clients and servers. Server is responsible for hosting, managing, and delivering services to clients. Clients are connected to the server and communicate with it over the internet using a computer network. So when a client needs a service, it sends a request to the server, which processes the request and sends a response back to the client.

Client server architecture example

Client-server architecture is used in many different types of applications. Here are two examples:

Healthcare application: A client computer will be running an application to enter patient information. At the same time, a server computer will be running another code to retrieve and manage database system.

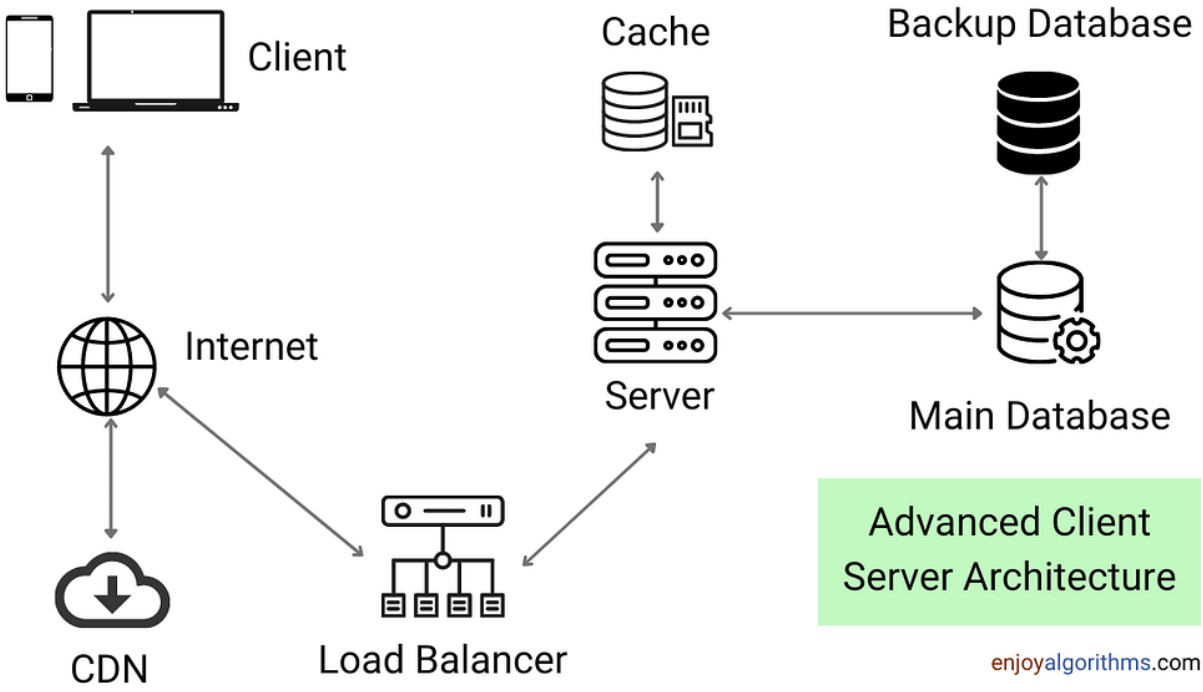
Banking application: When a bank customer accesses banking services with a web browser (client), web browser initiates a request to the webserver. Customer's login credentials will be stored in a database, and webserver accesses database server as a client. Now application server process the returned data by applying the business logic of banking application and provide output to the webserver. Finally, webserver returns the result to the client (web browser) for display.

Other applications that use the client-server architecture include email, the World Wide Web, and network printing.

Components of client server architecture

Client-server architecture consists of four components: client, load balancer, servers, and network-layer protocols.

- **Client** is a software application that requests resources and services made available by servers. It runs on a user's local computer or a remote machine and connects to a server. For example, a web browser is a type of client that sends requests to web servers.
- **Server** is a software program that receives and processes requests from clients. It operates on a remote machine and can be accessed by a user's local computer or workstation. A client can use a server to share resources and distribute tasks. For example, a web server is a type of server that receives and processes requests from web clients (e.g., web browsers).
- **Load balancer** is responsible for distributing incoming requests across a group of servers to manage traffic and optimize resource usage.
- **Network protocols** are used to facilitate communication between clients and servers in the client-server model. The typical TCP/IP protocol suite is used to communicate using a request-response messaging pattern. It divides application data into packets that networks can deliver and manages flow control. Once a connection is established using the TCP protocol, it is maintained until the client and server have completed the message exchange. The IP protocol is connectionless, meaning that each independent unit of data is unrelated to any other data unit and travels through the internet.



How does client server model work?

The data flow in a client-server architecture is unidirectional and forms a cycle. It begins when a client requests data from the server, and the server processes the request by querying a database for relevant data. The database then returns the queried data to the server, which processes it and sends it back to the client. Clients cannot communicate directly with each other.

Here is an example of how a client, such as a web browser, interacts with servers in a client-server architecture:

1. The user enters a website URL into the browser.
2. The browser sends a request to the DNS (Domain Name System) server to look up the IP address of the web server.
3. The DNS server sends the IP address of the web server to the browser.
4. The browser sends an HTTP/HTTPS request to the IP address of the web server.
5. The web server sends the necessary files for the website back to the browser.
6. The browser renders the files and displays the website.

Client-server architecture types

There are several types of client-server architectures, each with its own advantages and disadvantages. Here are some of the most common types of client-server architectures:

Two-Tier Architecture

There are only two components in two-tier architecture: client and server. Here client directly communicates with the server by sending a request, and the server responds with the data that the client requires. This architecture is simple and easy to set up, which makes it suitable for small applications. However, as the application grows in size and complexity, it will be less efficient and take longer to process requests.

To understand this better, you can think of the client as a customer who goes to a fast-food restaurant and directly places an order with the cook (server). The cook then prepares and serves the food to the customer. However, the process may become less efficient if there are many customers or if the customer wants a more complex meal.

Three-Tier Architecture

There are three components in a three-tier architecture: client, application server, and database server. Here client sends a request to the application server, application server processes the request and communicates with the database server to retrieve or update the data. This architecture works well for medium to large-scale applications and provides better scalability and fault tolerance.

To understand this architecture better, you can think of the client as a customer who goes to a restaurant and places an order with a waiter (application server). The waiter then communicates with the kitchen staff (database server) to get the required ingredients and prepare the food. This will help the restaurant system to serve efficiently because multiple waiters and kitchen staff can be added to serve more customers.

N-Tier Architecture

N-tier architecture is a system where there are multiple tiers, including presentation, application, and data tiers. Each tier is responsible for a specific function, and the tiers communicate with each other to perform the required tasks. This architecture is suitable for large and complex applications and provides better scalability and fault tolerance than the two-tier and three-tier architectures.

To understand this better, you can think of the client as a customer who visits a restaurant with a well-organized hierarchy of staff. Each staff member has a specific role, such as host (presentation tier), waiter (application tier), and kitchen staff (data tier). The host greets the customer and shows them to their table, the waiter takes their order and communicates with the kitchen staff to prepare the food, and the kitchen staff delivers the food to the table. This allows for a more efficient and effective process, as each staff member has a specific role and can focus on their responsibilities.

Client Server vs Peer to Peer Architecture

Here are some major differences between peer-to-peer and client-server architecture:

Client-Server Architecture	Peer-to-Peer Architecture
A clear separation between clients and servers.	No differentiation between clients and servers.
Data is provided only in response to a request.	Peers have the authority to request as well as provide a service.
Centralized data management.	It has own data and applications.
Purpose is to store and exchange information.	Goal is to maintain connections among peers.
Suitable for small as well as large networks.	Suitable for less number of users or devices.

Advantages of client server architecture

The client-server architecture has several benefits, including:

Centralized management: Client-server architecture involves storing and managing data in one central location. This allows for complete control over processes and activities and makes it easy to share resources and data across different platforms. Users also have the authority to access any file stored in the central storage at any time.

Flexibility: The data passed between the client and server and the services provided by the server are entirely at the discretion of the programmer. As a result, there are many ways to use the client-server architecture to solve future problems. Additionally, this architecture can be easily combined with other types of architectures on the client or server sides.

Extensibility: The system can be updated based on changes in functional and non-functional requirements without altering the client-server architecture or disrupting service.

Transparency: Clients only make requests to the server with their input

data, so they don't see how the server will handle the request. It may seem like a single, central server to the user.

Availability: Servers do not usually need to shut down or restart for long periods of time, so server uptime is possible during maintenance through server duplication. Additionally, there is a clear distinction between clients and servers, as clients consume services, and servers provide them. If multiple servers offer the same services, the system can still function even if one or more servers fail.

Scalability: The client-server architecture is capable of adding or removing servers in the network (horizontal scaling) or migrating to larger and faster server machines (vertical scaling).

Disadvantages of client server architecture

There are some potential drawbacks to the client-server architecture:

- Centralized control can lead to increased chances of failure. When many clients send simultaneous requests to the server, it can overload the server and slow down performance. This can also lead to server failure, causing the entire system to go down and clients not to receive any responses.
- Servers are more powerful than client computers, which means they are more expensive. They also require person with networking and infrastructure knowledge to manage the system.
- The client-server architecture is vulnerable to Denial of Service (DoS) attacks because the number of servers is typically smaller than the number of clients.
- Data packets may be modified during transmission, leading to the potential loss of useful information.

Use cases of client server architecture

- The client-server architecture is suitable for applications that require separation of concerns between the client and server and the ability

of computer systems or software to exchange and make use of information.
- It is often used in systems that require functional separation. Clients and servers each have their own set of responsibilities. Validation may be handled and managed on the client side, while the server is responsible for executing the client's request and returning the result. As a result, both the client and server can assist in implementing abstract functions without interfering with each other's capabilities.
- To address scalability issues, modern solutions have been developed within the client-server architecture, such as load balancing, sharding, and partitioning. The separation of functionality allows each layer to work more efficiently at a large scale.

Conclusion

We hope you enjoyed this blog on client-server architecture. We covered the fundamental concept behind it, how it works, and how it can be an efficient way of interaction. We hope you found it helpful. If you have any thoughts or feedback, please share them with us in the comments below. Thanks for reading!

If you have any queries/doubts/feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design, Enjoy algorithms!