Client Server Architecture

In general, what is a client server architecture?

- A centralized network architecture that classifies computer into two sections, client and server.
- A client is the requester, which can be a program that we use to make requests through the network with parameters included.
- A server is the response provider, which is a program that listens for the client's requests and responds to them.
- The server component provides a function or service to one or many clients, which initiate requests for such services.
- Server itself might be a client. For example, the server could request something from a database server, which in this case, would make the server a client of the database server.
- Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.

Have its own vocabulary for its components and connectors? (define)

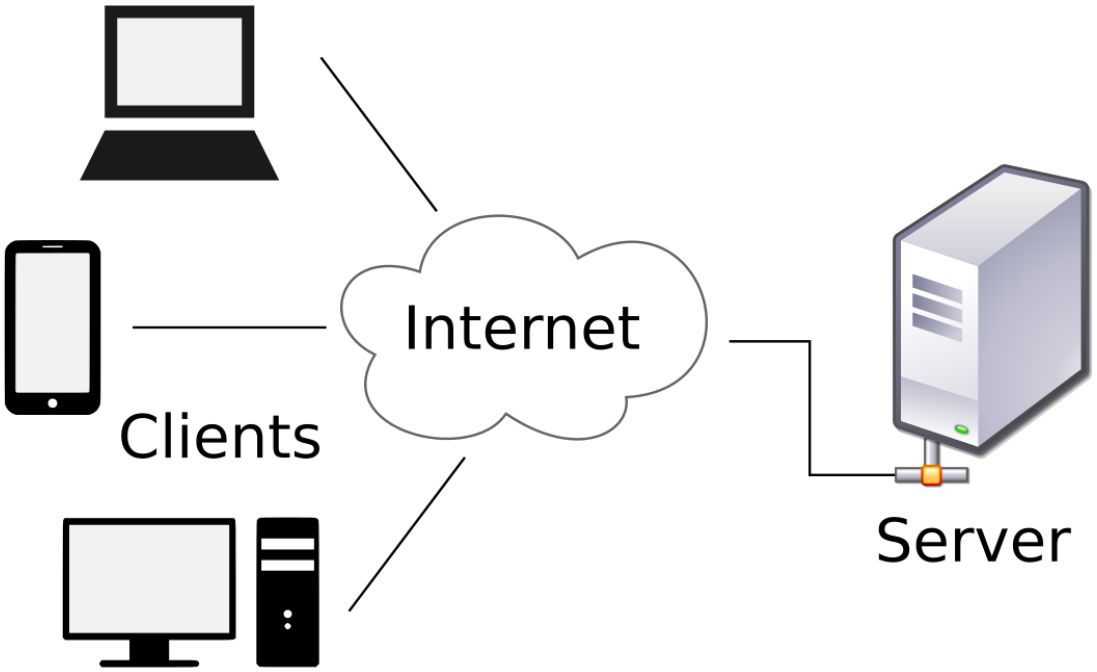
- **Components**:
  - The Server to listen for requests
  - The Clients to connect to the server
  - The Connection medium

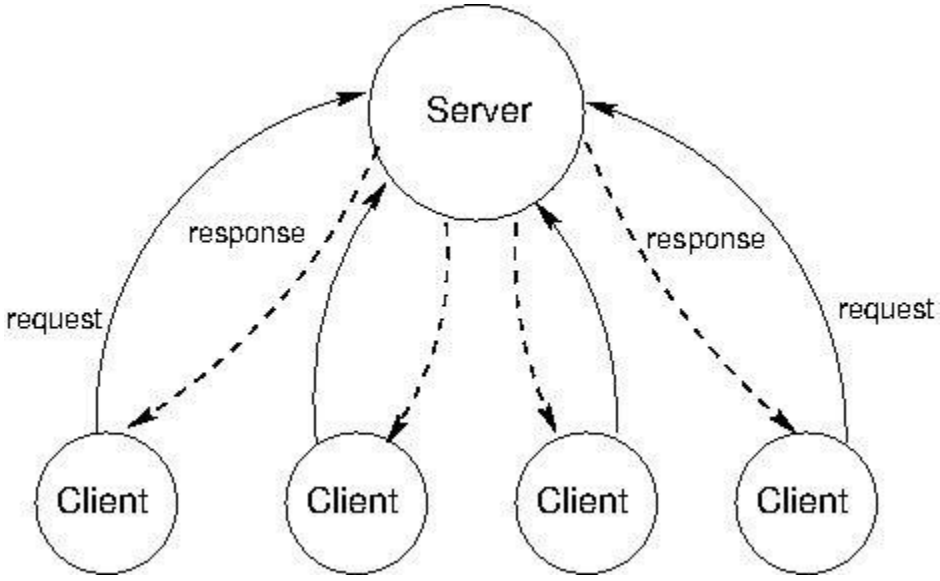
- **Connectors**: Protocols, Remote procedure calls (RPC)
  - Protocols: The special set of rules that end points in a telecommunication connection use when they communicate. Examples: TCP/IP, HTTP, FTP and etc.
  - RPC: when a computer program causes a procedure (subroutine) to execute in a different address space, which is coded as if it were a normal

(local) procedure call, without the programmer explicitly coding the details for the remote interaction.
    - Data is sent/received though connectors

Impose specific topological constraints? (diagram)

- Two levels, typically many clients with one server.
- From client to server and server to client.
- Constraints:
    - clients cannot communicate directly with each other. If needed, the server acts as a message relay for the clients to communicate.
    - Only clients can initiates communication
- All workloads are done at the server side.





Most applicable to specific kinds of problems?

- Where data can be centralized and easy to do for collaboration
- Where all clients are requesting the same type of data.
- Where clients can give specific information to request different data dynamically
- Less computational burden on the client side, which make client more lightweight.
- When clients are unable to do the heavy computation and the computation are done on the server side.
- Provide better data integrity and backup system, thus higher reliability.
- In general, people can access the data at anytime as long as they have network and authorization.
- When mobility is needed, applicated and data can be easily moved and replicated.

Engender specific kinds of change resilience (Advantage)?

- **Centralization of control:**
  - A dedicated server controls the access of resources and integrity of the data so that a program or unauthorized client cannot damage the system easily.
  - Changes only need to be done on the server and the clients will be able to receive
  - Network processing is done centrally, not at individual computers, which reduce the burden of the OS.

- **Scalability:**
  - You can increase the capacity of clients and servers separately. Any element can be increased or enhanced at any time, or you can add new nodes to the network.
  - You can add resources in the form of network segments, computers and servers to a client server network without major interruptions to the network.
  - Update task for data or other resources more efficient and easier to managed.

- **Easy maintenance:**
  - Since backup, security and antivirus are centralized, it is easier to setup and troubleshoot, where everything takes place at one physical server.
  - Fewer support staff are needed to manage centralized security accounts than would be needed if security and resource access had to be configured on each individual computer on the network.

Have any specific negative behaviours (Disadvantage)?

- **Single point of failure**: Since there's a reliance on the central server, if it fails, client requests cannot be done.
- **Traffic congestion:** Happens when a large number of simultaneous clients send requests to the same server. This might cause the server to slow down or even shut down.
- **Cost**: The cost of server hardware and software is much greater than the cost of buying desktop hardware and software licences. Thus it is expensive to scale or even hard to scale.

Support/inhibit specific NFPs?

- Complexity
  - Isolates the functionality to the server and the interaction to the client
  - Fairly cohesive
  - The topological hierarchy is very simple, only 2 nodes depth with server and client

- Scalability/Heterogeneity
  - Pros
    - Components are focused

        - Connectors are direct and simple
    - Cons
        - It is bottlenecked by the server
        - Does not replicate data(unless the model is extended by having multiple servers, load balancing...)
        - It is expensive to scale for more client connecting to the server and requires either more sophisticated hardware or more servers.
- Portability
    - The client can be ported and still use the same server
- Evolvability
    - Changes to the server processing does not affect the client
    - Changes to the client interface does not have to affect the server
    - No implicit connectors
- Dependability
    - Does support exception handling
    - Clients can connect and disconnect without affecting others
    - Only have to backup the server
- Reliability
    - A single point of failure. When the server is down, there will be no services to the client.

Skit Example Descriptions:

The examples will be a restaurant analogy with the waiter/restaurant as the server and the customers as the clients.

General case:

Scenario 1: General case

There are 2 customers ordering food and the waiter takes their orders separately and brings them back the food they ordered. This example demonstrates how the server can be connected to more than one client. Furthermore, it also shows that for a communication to happen between a client and a server, the client have to be the one to first initiates the request after which the server will reply with a response.

Scenario 2: Client communication

A customer orders food and tells the server that he will pay for the other customer's bill. This demonstrates that if the clients want to communicate with each other, they have to do it through the server.

Advantages:

Scenario 3: Multiple client types with the same server
In this scenario, a customer orders delivery on the phone. This shows that there can be multiple types of clients, but still having the same server.

Scenario 4: server update
Here, the server is updated by having a different menu, the customers can still interact with the server the same as before.
Server update

Disadvantage:

Scenario 5: Server single point of failure
The server crashed and is unable to tend to the customers. This example demonstrates the single point of failure problem that the client server architecture has, where everything is reliant on the one server.

Scenario 6: Denial of service/ server bottleneck
This scenario has one customer constantly making requests to the server, inhibiting the other customers from making requests. This shows that a malicious client can overload the server by making it do multiple or difficult requests. This also demonstrates that the server is a potential bottleneck of the whole system if not enough resources are dedicated to it.