**Client Server**

1. **Tell us about the features of client/server**.

**Client-server architecture**, architecture of a computer network in which many [clients](https://www.britannica.com/technology/client) (remote processors) request and receive service from a centralized [server](https://www.britannica.com/technology/server) (host computer). Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns. Servers wait for requests to arrive from clients and then respond to them. Ideally, a server provides a standardized transparent interface to clients so that clients need not be aware of the specifics of the system (i.e., the hardware and software) that is providing the service. Clients are often situated at workstations or on personal computers, while servers are located elsewhere on the network, usually on more powerful machines. This computing model is especially effective when clients and the server each have distinct tasks that they routinely perform.

1. **What is a Web server in a client server environment?**

"Web server" can refer to hardware or software, or both of them working together.

* On the hardware side, a web server is a computer that stores web server software and a website's component files (e.g. HTML documents, images, CSS stylesheets, and JavaScript files). It is connected to the Internet and supports physical data interchange with other devices connected to the web.
* On the software side, a web server includes several parts that control how web users access hosted files, at minimum an *HTTP server*. An HTTP server is a piece of software that understands [URLs](https://developer.mozilla.org/en-US/docs/Glossary/URL) (web addresses) and [HTTP](https://developer.mozilla.org/en-US/docs/Glossary/HTTP) (the protocol your browser uses to view webpages). It can be accessed through the domain names (like mozilla.org) of websites it stores and delivers their content to the end-user's device.

1. **What is the role of the presentation layer**

This tier, which is built with HTML5, cascading style sheets (CSS) and Javascript, is deployed to a computing device through a web browser or a web-based application. The presentation tier communicates with the other tiers through application program interface (API) calls

1. **They say this architecture is secure, how is it done in your opinion?**

To enhance the system security of the client computers, Virtual Machine Monitor (VMM) and the Security Manager (SM) were designed and used to form the trusted platform and check the Operating System's integrity outside the Virtual Machine (VM). Plus, a distinct hardware channel and I/O device, the secure I/O, was used to prevent low level attacks, like keyboard sniffers, screen scrapers, and so on.

To improve the reliability of the critical servers, an infrastructure was designed to dynamically construct and configure server farms using a peer-to-peer architecture. The decentralized, self-organization and scalability characteristics of peer-to-peer architecture ensure that the system will have significantly low management cost, while providing highly reliable services in spite of attacks.

1. **What is a Database Server in a client server environment?**

A database server is responsible for database storage, access and processing in a client server environment.

1. **What are Super servers in client server environments?**

A super-server starts other servers when needed, normally with access to them checked by a TCP wrapper. It uses very few resources when in idle state. This can be ideal for workstations used for local web development, client/server development or low-traffic daemons with occasional usage (such as ident and SSH).

1. **Explain 2-Tier and 3-Tier architecture**

**2-Tier Architecture:**

This is where you have direct communication between a **client** and a **server**with no intermediary. It is divided into two parts: client application and a database.

The client in a Two-tier**architecture** application has the code written for saving data in the database. The client sends a request to the server, where it then processes the request and sends back the data. Meaning the client handles both the Presentation layer (application interface) and Application layer (logical operations), while the server system handles the database layer.

**3-Tier Architecture:**

The **Three-tier Architecture** is divided into 3 parts

1. Presentation Layer (Client tier)
2. Application Layer (Business tier)
3. Database Layer (Data tier)

The Presentation Layer is the topmost layer of an application. This is the layer seen when using the software(Interface, web pages). By using the software we access web pages. Its main function is to communicate with the application layer. This layer passes information which is given by the user in terms of keyboard actions and mouse clicks to the application layer.

The Application Layer is also known as the Business logic layer. Here’s where we find logic controls and functionality that processes data received from the presentation layer and database layer. It acts as an intermediary between the presentation and database layer.

The Database Layer is the layer that stores data with the retrieval storage and execution methods made by the application layer. It contains methods that connect to the database and performs the required actions needed. These are Insert, update or delete.

1. **What is a File server?**

A file server is a server that provides access to files. It acts as a central file storage location that can be accessed by multiple systems. File servers are commonly found in enterprise settings, such as company networks, but they are also used in schools, small organizations, and even home networks.

**SOA & MicroServices**

1. **What are the main benefits of SOA?**

* Services are reusable
* Services are easily maintained : One of the advantages of SOA is the elimination of the complexities associated with interacting services within an environment. When it comes to updating, upgrading, and maintaining the services in the SOA environment there are no complications resulting from interactions with other connected and interacting services. This is because the program is complete and self-contained. In environments with complex components, this makes things easy.
* Reliability: SOA services are complete and self-contained programs. This makes it easy for testing, debugging or any form of maintenance.
* Availability: Normally, services in a SOA are available to any requester that needs them. For this reason, SOA services are made available via public directories where they can be searched for and discovered. Consumers, therefore, are always able to access the services.
* Scalability: Different organizations have different levels of complexity in their architecture. So, the ability of services to be successfully run on different servers within an environment increases the scalability and availability of the service.
* High Quality Services: Servers, as we have seen, may have many applications to which they may be applied. As such, there will be no functional redundancy. Data is always consistent, and the service codes are constantly maintained resulting in continuous high-quality applications.
* Platform Independence: One of the main advantages of SOA services is their public availability to any system that needs them, independent of the platform on which the system runs. This platform independence allows the integration of different requested services from different vendors to run harmoniously, regardless of the platform.
* Increased Productivity: The world is constantly changing, and technology is changing and evolving, trying to keep up. Services that have been versatile in their applications need to be constantly improved. Developers using the concepts of previous services need not create every application from scratch but can adapt and reuse old concepts and techniques when evolving into new services and technology.

1. **How can you achieve loose coupling in SOA**

There are a couple of approaches you can take. Our SOA architecture involves XML messages sent to and from the services. One way we achieve what you describe is by avoiding the use of a data binding library to our XML schema and use a generic XML parser to get just the data nodes you want ignoring those you aren't interested in. This way the service can add additional new nodes to the message without breaking anyone currently using it. We typically only do this when we need just one or two pieces of information from a larger schema structure.

Alternatively, the other (preferred) solution we use is versioning. A version of a service adheres to a particular schema/interface. When the schema changes (e.g. the interface is extended or modified), we create a new version of the service. At any time, we may have 2 or 3 versions on the go at any one time. In time, we deprecate and then remove older versions, while eventually migrating dependent code onto newer versions. This way those dependent on the service can continue using the existing version of the service while some particular dependency can 'upgrade' to the new version. Which versions of a service are called are defined in a configuration file for the dependent code. Note that it is not only the schema which gets versioned, but all of the underlying implementation code as well.

1. **Are web services and SOA the same?**

SOA is an architectural concept while web services are used to complete them. Web services are the preferred standards that are satisfied to achieve the architectural specifications of SOA. When one uses SOA all services need to be loosely coupled. Also, SOA services should be able to describe themselves and WSDL services will be describing how we can access the services. Also, these services can be accessed through WSDL. SOA services are located in a directory and here UDDI describes where these web services can be found.

1. **What is a reusable service?**

Reusable service is a stateless functionality that has the required granularity. It can be a part of a composite application or composite server. A reusable service should be identified with any activity prescribed by business and which has its specifications. A service constraint may be security, QoS, SLA or any usage policies. It may be defined by different runtime contracts, multiple interfaces, and different implementations. A reusable service is looked over by enterprise-level throughout its lifecycle, starting from design time through its runtime. Its reuse should also be promoted through a pre-defined process and its reuse can be measured.

1. **What are the disadvantages of SOA?**

* Extra Overload: In SOA, all inputs are validated before it is sent to the service. If you are using multiple services, then it will overload your system with extra computation.
* High Cost: SOA is costly in terms of human resource, development, and technology.
* High Bandwidth Server: As some web service sends and receives messages and information frequently so it easily reaches a million requests per day. So, it involves a high-speed server with a lot of data bandwidth to run a web service.

1. **What is ESB and where does it fit in?**

ESB stands for Enterprise Service Bus. Unlike other relationships, it provided any to any connectivity between different companies. Also, you may need to consider deployment services, IT services, etc. The SOA architecture enables SOA to meet all life’s priorities. The ESB is part of this reference architecture and provides the backbone of an SOA but it should not be considered an SOA by itself.

1. **In SOA do we need to build a system from scratch?**

No, if we need to integrate any existing system you just can loosely couple wrappers which help in wrapping all customer services and expose all functionalities in a generic manner.

1. **What is the most important skill needed to adopt SOA? technical or cultural?**

Surely cultural. SOA does require people to think of business and technology differently. Instead of thinking of technology first (e.g., If we implement this system, what kinds of things can we do with it?), practitioners must first think in terms of business functions, or services (e.g., My company does these business functions, so how can I set up my IT system to do those things for me most efficiently?).

It is expected that adoption of SOA will change business IT departments, creating service-oriented (instead of technology-oriented) IT organizations.

1. **List down the advantages of Microservices Architecture.**

* Better fault isolation; if one microservice fails, the others will continue to work.
* Code for different services can be written in different languages.
* The microservice architecture enables continuous delivery.
* Easy to understand since they represent a small piece of functionality, and easy to modify for developers, thus they can help a new team member become productive quickly.
* The code is organized around business capabilities.
* Scalability and reusability, as well as efficiency. Easy to scale and integrate with third-party services.
* Components can be spread across multiple servers or even multiple data centers.
* Complement cloud activities.
* Microservices simplify security monitoring because the various parts of an app are isolated. A security problem could happen in one section without affecting other areas of the project.
* Increase the autonomy of individual development teams within an organization, as ideas can be implemented and deployed without having to coordinate with a wider IT delivery function.

1. **What are the best practices to design Microservices?**

* Create a Separate Data Store for Each Microservice
* Keep Code at a Similar Level of Maturity
* Do a Separate Build for Each Microservice
* Deploy in Containers
* Treat Servers as Stateless

1. **How does Microservice Architecture work?**
2. **What are the pros and cons of Microservice Architecture?**

**Pros:**

* Greater agility
* Better scalability
* Faster development cycles (easier deployment and debugging)
* Easier to create a CI/CD pipeline for single-responsibility services
* Isolated services have better fault tolerance
* Platform- and language agnostic services
* Cloud-readiness
* Faster time to market

**Cons:**

* Needs more collaboration (each team has to cover the whole microservice lifecycle)
* Harder to test and monitor because of the complexity of the architecture
* Poorer performance, as microservices need to communicate (network latency, message processing, etc.)
* Harder to maintain the network (has less fault tolerance, needs more load balancing, etc.)
* Doesn’t work without the proper corporate culture (DevOps culture, automation practices, etc.)
* Security issues (harder to maintain transaction safety, distributed communication goes wrong more likely, etc.)

1. **What is the difference between Monolithic, SOA and Microservices Architecture?**
2. **What are the challenges you face while working Microservice Architectures?**

* Data synchronization (Consistency)
* Security
* Services Communication
* Data Staleness
* Testing: Testing is much more complex in a microservices environment due to the different services, their integration, and interdependencies. The team members responsible for quality assurance need to be knowledgeable on the order and channels of communications between services to have full coverage in their test cases. The asynchronous aspect of microservices also makes it harder to test in lower environments. Indistinct behaviors from microservices are harder to predict and validate.
* Monitoring and Performance : The traditional forms of monitoring and diagnostics will not align well with microservices since you have multiple services making up the same functionality previously supported by a single application. When a problem arises in the application, finding the root cause can be challenging if you do not have a means of monitoring and tracking the path a specific request took, like how many and which microservices were traversed for a specific request coming from a user interface.
* Embracing DevOps culture: Separate teams need agility, autonomy, and continuous delivery to be able to deliver initial releases and subsequent iterative changes. A lack of DevOps culture can bottle up releases and impact the overall time to market and the response to business requests and issues.
* Fault Tolerance: It is important that individual services do not bring down the overall system. Fault tolerance at the service level, and more importantly, at the overall solution level, is critical. Given the complexity of a microservices environment and the complex dependency chains, failure is inevitable. Microservices need to be able to withstand both internal and external failures. Robust resiliency testing is key to successful issue preparedness.
* Design with failure in mind: While this is counter-intuitive to many, expecting failure scenarios and building a robust set of microservices is imperative to a successful implementation. When more failure situations are predicted during design, the more exception handling mechanisms will be built, and seamless resolution of issues will be handled better. This is easier said than done.
* Cyclic dependencies: Dependency management across different services and their functionality is very important and cyclic dependencies can be a headache if not identified and resolved promptly. In microservice architecture, you’re even more vulnerable to errors coming from dependency issues. Decisions made around upgrades on related services with these dependencies are critical.

1. **What are the characteristics of Microservices?**

* Microservice architectural style is an approach to developing a single application as a suite of small services.
* Services are built around business capabilities, independently deployable and packaged, each running in its own process.
* Each Service should have separate database layer.
* Each Service can have independent codebase, CI/CD tooling sets.
* Each Service can be tested in isolation without dependent on other services
* Each service can pick the best technology stack for its use cases (no need to stick into one framework throughout the entire application).
* Each Service should have monitoring and troubleshooting capabilities for operation team
* Services should implement Retry functionality in case of network failure or system failure
* Each Service can implement independent security mechanism
* The main difference between SOA and Microservices is that a microservice employs a practice that attempts to eliminate any dependencies on other microservices. SOA does not make this  
  practice explicit as a requirement.
* Services can use HTTP(Rest) or messaging for communication or any other lightweight communication protocol.
* Well understood Distribution Transaction management
* Presenting API
* Clean and Clear Separation of Stateless and Stateful Services
* Do Not Share Libraries or SDKs
* Each Service can be run without waiting for other service to go online
* Implement a Self-Registration and Discovery Mechanism
* Service can use different language, framework and technologies
* Maintain Independent Revisions and Build Environments to maintains compatibility with other services.
* An architect can explicitly check for rules and constraints.