CLIENT-SERVER ARCHITECTURE

* In client server computing, the clients requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server. Both the client and server usually communicate via a computer network but sometimes they may reside in the same system.
* Both entities should follow a common protocol
* There can be multiple layers and tiers in CSA.
* server can only accommodate a limited number of client requests at a time. So it uses a system based to priority to respond to the requests
* An example of a client server computing system is a web server. It returns the web pages to the clients that requested them
* Server: Robot type programs that constantly run and exchange information with remote users. Eg: Application Server like SAP, Database Server like MySql Server etc.
* Client: program that access and exchange information with servers. Eg: ARC,postman etc.

WEB SERVER IN CLIENT SERVER ENVIRONMENT

* A **web server** is server software, or a system of one or more computers dedicated to running this software, that can satisfy client HTTP requests on the public World Wide Web or also on private LANs and WANs
* The primary function of a web server is to store, process and deliver web contents/web resources to clients

ROLE OF THE PRESENTATION LAYER

* It’s a layer in the 3 tier CSA which bears the function to receive input and display output
* Top most level of the application
* Provides the applications’ user interface
* The presentation tier displays information related to such services as browsing merchandise, purchasing, and shopping cart contents

SECURITY IN 3 TIER-CSA

* Imagine if there is only one server for all functionalities. When some sort of external intrusion/hacking happens to the system, we may face partial or complete data loss
* But in the case of a 3 tier CSA, we can provide firewalls in between the tiers, so that the intruder finds it difficult to access the underlying raw data inside the database tier

DATABASE SERVER IN 3 TIER CSA

* All the raw data of the system resides inside the database server
* It stores and manages data
* As it contains sensitive information, it cannot be exposed directly to a cient. Some sort of firewall mechanism is needed in between presentation tier and databse tier
* It can be a RDBMS like MySQL or a NoSQL database like mongodb

SUPER SERVERS IN CSA

* A **super-server** or sometimes called a **service dispatcher**  is a type of daemon run generally on Unix-like systems.
* 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

2-TIER AND 3-TIER ARCHITECTURE

* 2-tier is a CSA whereas 3-tier is a web based application
* In 2-tier, the application tier is buried inside the presentation tier or database tier. In 3-tier, the application tier resides as a middleware, seperated from both presentation tier and database tiers
* 2-tier is comparatively easy to maintain, but slower, less secure and have lesser scalability than 3-tier.

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.
* A file server may be a dedicated system, such as network attached storage ([NAS](https://techterms.com/definition/nas)) device, or it may simply be a computer that hosts shared files

MAIN BENEFITS OF SOA

* Service Reusability: service contains agnostic logic that can be positioned as reusable enterprise resource
* Easy maintenance: saves money, enhances quality
* Platform independent: a client from any device using any OS in any language can use the service
* Availability: usually obtained by having several instances of services running simultaneously. If one or more service instances fail, requests can still be serviced by the other instances.
* Reliability: Easy to debug and troubleshoot
* Scalability: It can easily grow onto multiple servers and across datacenters.

LOOSE COUPLING IN SOA

* Can be achieved through the use of **Enterprise Service Bus (ESB)**
* This approach helps to coordinate and arrange the different elements that are in the form of distributed services on a network. This approach considers the systems to be discrete and distributed services that connect to one another through message oriented infrastructure that is asynchronous. This kind of a message-oriented infrastructure makes it possible to have loosely coupled connections between independent services or modules

ARE WEB SERVICES AND SOA SAME?

* SOA is an architectural concept which focus on having different services communicating with each other to carry out a bigger job
* A web service is a basic building block in a SOA. When multiple services are combined, we have an application that falls under SOA

WHAT IS A REUSABLE SERVICE?

* A service which reflects agnostic logic that can be positioned as a reusable enterprise resource
* Service logic can be repeatedly leveraged so as to achieve a high ROI
* Leads to increased business agility in an organization
* Enable the creation of business inventories that can be easily integrated and used in various use-cases

DISADVANTAGES OF SOA

* Large investment: SOA is a great choice for future business developments. It allows working on different applications simultaneously, but the implementation part is costly.
* Greater load and increased response time: Each interaction between services demands a validation of input parameters, which increases the load. This leads to increased response time
* Vast variety of services: the services exchange messages during each communications. These messages often gets overwhelming leading to a messed up management system

WHAT IS ESB? WHERE DOES IT FIT IN?

* An enterprise service bus (ESB for short) refers to software architecture that allows for the integration of enterprise applications and services, such as middleware infrastructure platforms
* Like a router, It provides the connections between applications that need to communicate with one another. Businesses use ESBs in enterprise application integration.
* Interactions take place across the Bus
* It came into popularity as the point to point architecture increases system complexity, limits reusability and is monolithic. P2P architecture no more offers modern business functionalities

IN SOA, DO WE NEED TO BUILD A SYSTEM FROM SCRATCH?

* No. If you need to integrate or make an existing system as a business service, you just need to create loosely coupled wrappers which will wrap your custom systems and expose the systems functionality in a generic fashion to the external world.

WHAT IS THE MOST IMPORTANT SKILL NEEDED TO BUILD A SOA? TECHNICAL OR CULTURAL?

* Surely cultural. Instead of a technical point of view, practitioners must think in terms of business or services.(How can I reorient the current IT system to realize our functions efficiently?)

ADVANTAGES OF MICROSERVICE ARCHITECTURE

* Agility: Microservices support agile development. Any new feature can be quickly developed and discarded again..
* Small,focused team: microservices can be assigned to specific development teams, which allows them to focus solely on one service or feature. This means teams can work autonomously without worrying what’s going on with the rest of the app.
* Smaller codebase: enables team to understand the code better, leading to better maintenance and deployability
* Mix of technology:  Different languages and technologies can be used to build different services of the same application.
* Fault Isolation: even if one service of the application does not work, the system still continues to function.
* Granular Scalability: Individual components can scale as per need, there is no need to scale all components together.
* Data Isolation: Allows each service to be managed independently

WHAT ARE THE BEST PRACTICES TO DESIGN A MICROSERVICE?

* Have separate data storages: make the data private to each of the microservices to avoid coupling. Use architectural pattern like CQRS
* Build dedicated team: allocate dedicated teams for every microservice if you are planning to scale linearly and efficiently
* Use automation for individual deployment: With automation, microservices can be wrapped in containers and efficiently deployed to any environment, including the cloud. It reduces the lead time
* Leverage the benefits of REST API: Developers need not install any additional software or libraries while creating a REST API. At the same time, they provide a great deal of flexibility since the data is not tied to any particular method or resource
* Understand the cultural shift:  Developers need to be sensitized with the expectations of the new working environment and how the long term vision of the company impacts their day to day working capacities.
* Breakdown the migration to steps: Once you have enough new services in place (and the teams have been sensitized about the new processes) figure out how you can break down the old architecture into relevant components and begin migrating them one by one.
* Build the Splitting System Right into the Mix: inspect the monolithic structure to understand the gaps it has and components that are causing the most trouble and then go on to transform this part into a microservice.
* Isolate the Runtime Processes: You need to implement some form of distributed computing to pull this off from a pool of possible choices. Do we need to adopt containerization, event architectures, various HTTP management approaches, service meshes, and circuit breakers? Figure this out first
* Pair the Right Technology with the Right Microservice: Before selecting the technology to be used, consider the parameters like maintenability, fault tolerance,scalability,cost,ease of deployment etc.
* Consider Using Domain-Driven Design: It is nothing more than Object Oriented Programming applied to business models. It is a type of design principle that makes use of practical rules and ideas to express an object-oriented model
* Distinguish Between Dedicated and On-Demand Resources: For a superior customer experience. For example, an e-commerce platform that builds its microservices and cloud architecture in ways that quickly (and securely) moves workloads between its on-premise and cloud environments. Not only does it increase the response time, but it also makes migrating to a cloud-based working environment much more intuitive
* Govern the Dependency on Open Source Tools:  ensure that they are not over-relied upon in ways that interfere with the performance or security of the architecture.

HOW DOES MICROSERVICE ARCHITECTURE WORKS?

* Microservices are a popular software design architecture that breaks apart monolithic systems. Applications are built as collections of loosely coupled services. Each microservice is responsible for a single feature. They interact with each other through communication protocols such as HTTP and TCP
* A typical Microservice Architecture (MSA) should consist of the following components:
  + - Clients
    - Identity Providers
    - API Gateway
    - Messaging Formats
    - Databases
    - Static Content
    - Management
    - Service Discovery
* **1. Clients:** The architecture starts with different types of clients, from different devices trying to perform various management capabilities such as search, build, configure etc
* **2. Identity Providers:** These requests from the clients are then passed on the identity providers who authenticate the requests of clients and communicate the requests to API Gateway. The requests are then communicated to the internal services via well-defined  API Gateway
* **3. API Gateway:** Since clients don’t call the services directly, API Gateway acts as an entry point for the clients to forward requests to appropriate microservices.
* **4. Messaging Formats:** There are two types of messages through which they communicate: Asynchronous messaging and synchronous messaging
* **5. Data Handling:** Each Microservice owns a private database to capture their data and implement the respective business functionality.Also, the databases of Microservices are updated through their service API only
* **6. Static Content:** After the Microservices communicate within themselves, they deploy the static content to a cloud-based storage service that can deliver them directly to the clients via **Content Delivery Networks (CDNs)**.
* **7. Management:** This component is responsible for balancing the services on nodes and identifying failures
* **8. Service Discovery:** Acts as a guide to Microservices to find the route of communication between them as it maintains a list of services on which nodes are located.

PROS AND CONS OF MICROSERVICE ARCHITECTURE

* PROS: Freedom to use different technologies, Each microservice focuses on single business capability, Supports individual deployable units, Allows frequent software releases, Ensures security of each service, Multiple services are parallelly developed and deployed
* CONS: Increases troubleshooting challenges, Increases delay due to remote calls, Increased efforts for configuration and other operations, Difficult to maintain transaction safety, Tough to track data across various service boundaries, Difficult to move code between services

DIFFERENCE BETWEEN MONOLYTHIC,SOA AND MICROSERVICE ARCHITECTURE

* The concept of **monolithic software** lies in different components of an application being combined into a single program on a single platform. Usually, a monolithic app consists of a database, client-side user interface, and server-side application. All the software’s parts are unified and all its functions are managed in one place
  + PROS: simpler development and deployment,fewer cross cutting concerns,Better performance
  + CONS: code gets cumbersome over time,difficult to adopt new technologieslimited agility
* A **service-oriented architecture** (SOA) is a software architecture style that refers to an application composed of discrete and loosely coupled software agents that perform a required function. SOA has two main roles: a service provider and a service consumer. Both of these roles can be played by a software agent. The concept of SOA lies in the following: an application can be designed and built in a way that its modules are integrated seamlessly and can be easily reused.
  + PROS: reusability of services,better maintainability,higher reliability,parallel development
  + CONS: complex management, high investment costs, extra overload
* **Microservice** is a type of service-oriented software architecture that focuses on building a series of autonomous components that make up an app. Unlike monolithic apps built as a single indivisible unit, microservice apps consist of multiple independent components that are glued together with APIs
  + PROS: easy to develop,test and deploy, increased agility,ability to scale horizontally
  + CONS: complexity,security concerns, different programming languages

CHALLENGES WE FACE WHILE WORKING WITH MICROSERVICE ARCHITECTURE

* Managing microservices: As the number of microservices increases, managing them gets more challenging. It is important that management is planned before or while microservices are being built
* Monitoring: 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
* 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: 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.
* 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.
* 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

CHARACTERISTICS OF MICROSERVICES

* Componentization via services
* Organized around business capabilities
* Products not projects
* Smart endpoints and dump pipes
* Decentralized governance
* Decentralized data management
* Infrastructure automation
* Design for failure
* Evolutionary design

.