**The CAP Theorem for distributed computing** was published by Eric Brewer. This states that it is not possible for a distributed computer system to simultaneously provide all three of the following guarantees:

1. *Consistency* (all nodes see the same data even at the same time with concurrent updates )
2. *Availability* (a guarantee that every request receives a response about whether it was successful or failed)
3. *Partition tolerance* (the system continues to operate despite arbitrary message loss or failure of part of the system)

Scalability is the ability of a system, network, or process to handle a growing amount of load by adding more resources. The adding of resource can be done in two ways

* **Scaling Up**  
  This involves adding more resources to the existing nodes. For example, adding more RAM, Storage or processing power.
* **Scaling Out**  
  This involves adding more nodes to support more users.

**Availability** means the ability of the application user to access the system, If a user cannot access the application, it is assumed unavailable. High Availability means the application will be available, without interruption. Using redundant server nodes with clustering is a common way to achieve higher level of availability in web applications.

Availability is commonly expressed as a percentage of uptime in a given year.

Clustering is needed for achieving high availability for a server software. The main purpose of clustering is to achieve 100% availability or a zero down time in service. A typical server software can be running on one computer machine and it can serve as long as there is no hardware failure or some other failure. By creating a cluster of more than one machine, we can reduce the chances of our service going un-available in case one of the machine fails.

ACID

* **Atomicity**: It ensures all-or-none rule for database modifications.
* **Consistency**: Data values are consistent across the database.
* **Isolation**: Two transactions are said to be independent of one another.
* **Durability**: Data is not lost even at the time of server failure.

*ACID* is a acronym which is commonly used to define the properties of a relational database system, it stand for following terms

* **Atomicity** - This property guarantees that if one part of the transaction fails, the entire transaction will fail, and the database state will be left unchanged.
* **Consistency** - This property ensures that any transaction will bring the database from one valid state to another.
* **Isolation** - This property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially.
* **Durable** - means that once a transaction has been committed, it will remain so, even in the event of power loss.
* A **mutex** (or Mutual Exclusion Semaphores) is a **locking mechanism** used to synchronize access to a resource. Only one task (can be a thread or process based on OS abstraction) can acquire the mutex. It means there will be ownership associated with mutex, and only the owner can release the lock (mutex).
* **Semaphore** (or Binary Semaphore) is **signaling mechanism** (“I am done, you can carry on” kind of signal). For example, if you are listening songs (assume it as one task) on your mobile and at the same time your friend called you, an interrupt will be triggered upon which an interrupt service routine (ISR) will signal the call processing task to wakeup. A binary semaphore is NOT protecting a resource from access. Semaphores are more suitable for some synchronization problems like producer-consumer.
* **Concurrency** is when two or more tasks can start, run, and complete in overlapping time **periods**. It doesn't necessarily mean they'll ever both be running **at the same instant**. For example, multitasking on a single-core machine.
* **Parallelism** is when tasks literally run at the same time, e.g., on a multicore processor.

Some of the advantages of web services are:

* **Interoperability**: Web services are accessible over network and runs on HTTP/SOAP protocol and uses XML/JSON to transport data, hence it can be developed in any programming language. Web service can be written in java programming and client can be PHP and vice versa.
* **Reusability**: One web service can be used by many client applications at the same time.
* **Loose Coupling**: Web services client code is totally independent with server code, so we have achieved loose coupling in our application.
* **Easy to deploy and integrate**, just like web applications.
* **Multiple service versions** can be running at same time.
* **Monolithic Architecture** is similar to a big container wherein all the software components of an application are assembled together and tightly packaged.
* A **Service-Oriented Architecture** is a collection of services which communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity.
* **Microservice Architecture** is an architectural style that structures an application as a collection of small autonomous services, modeled around a business domain.

There are plenty of pros that are offered by Microservices architecture. Here are a few of them:

* Microservices can adapt easily to other frameworks or technologies.
* Failure of a single process does not affect the entire system.
* Provides support to big enterprises as well as small teams.
* Can be deployed independently and in relatively less time.

The architectural style for creating web api are

* HTTP for client server communication
* XML/JSON as formatting language
* Simple URI as the address for the services
* Stateless communication
* **Windows Communication Foundation** is designed to exchange standard SOAP-based messages using variety of transport protocols like HTTP, TCP, NamedPipes or MSMQ, etc.
* On the other hand, **ASP.NET API** is a framework for building non-SOAP based services over HTTP only.

Some of the disadvantages of REST are:

* Since there is no contract defined between service and client, it has to be communicated through other means such as documentation or emails.
* Since it works on HTTP, there can’t be asynchronous calls.
* Sessions can’t be maintained.
* Developers practicing **continuous integration** merge their changes back to the main branch as often as possible. By doing so, you avoid the integration hell that usually happens when people wait for release day to merge their changes into the release branch.
* **Continuous delivery** is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that on top of having automated your testing, you also have automated your release process and you can deploy your application at any point of time by clicking on a button.
* **Continuous deployment** goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.
* A **clustered index** is a special type of index that reorders the way records in the table are physically stored. Therefore table can have only one clustered index. The leaf nodes of a clustered index contain the data pages.
* A **non clustered index** is a special type of index in which the logical order of the index does not match the physical stored order of the rows on disk. The leaf node of a non clustered index does not consist of the data pages. Instead, the leaf nodes contain index rows.