***What is scalability?***

*In software, scalability means a system’s ability to handle varying workloads as the needs of your business change. A scalable app is an app that remains productive whether it’s handling a lot of traffic, users, and data or a little. It doesn't matter whether you're adding a new functionality or handling a sudden surge in user requests, a scalable app should cope with all kinds of situations.*

*Scalability refers to the ability of a system to handle a growing amount of work or its potential to accommodate growth.*

* *A system that scales well is one that can maintain or even improve its performance, efficiency, and reliability when its workload or scope increases.*
* *Scalability is crucial for ensuring that a system can handle increased user traffic, data volume, or computational demands without experiencing a significant decrease in performance or requiring a complete redesign.*

***Importance of Scalability in System Design*** *–*

* *Handle Growth*
* *Improve Performance*
* *Ensure Availability*
* *Optimize Costs*
* *Support Innovation*

***Software scalability types*** *–*

*Horizontal software scalability (scaling out) - You can scale software horizontally by incorporating additional nodes into the system to handle a higher load, as it will be distributed across the machines. For instance, if an application starts experiencing delays, you can scale out by adding another server.*

*Horizontal scalability is a better choice when you can’t estimate how much load your application will need to handle in the future. It’s also a go-to option for software that needs to scale fast with no downtime.*

*Benefits:*

* *Resilience to failure. If one node fails, others will pick up the slack.*
* *There is no downtime period during scaling as there is no need to deactivate existing nodes while adding new ones.*
* *Theoretically, the possibilities to scale horizontally are unlimited.*

*Limitations:*

* *Added complexity. You need to determine how the workload is distributed among the nodes. You can use Kubernetes for load management.*
* *Higher costs. Adding new nodes costs more than upgrading existing ones.*
* *The overall software speed might be restricted by the speed of node communication.*

*Vertical software scalability (scaling up)*

*Vertical scalability is about adding more power to the existing hardware. Here you will update the existing server by adding more processing power, memory, etc. Another option is removing the old server and connecting a more advanced and capable one instead. This scalability type works well when you know the amount of extra load that you need to incorporate.*

*Benefits:*

* *There is no need to change the configuration or an application’s logic to adapt to the updated infrastructure.*
* *Lower expenses, as it costs less to upgrade than to add another machine.*

*Limitations:*

* *There is downtime during the upgrading process.*
* *The upgraded machine still presents a single point of failure.*
* *There is a limit on how much you can upgrade one device.*

***When do you absolutely need scalability*** *–*

*When software scalability is not needed:*

* *If the software is a proof of concept (PoC) or a prototype*
* *When developing internal software for small companies used only by employees*
* *Mobile/desktop app without a back end*

*For the rest, it’s strongly recommended to look into scalability options to be ready when the time comes. And how do you know it’s time to scale? When you notice performance degradation. Here are some indications:*

* *Application response time increases*
* *Inability to handle concurrent user requests.*
* *Increased error rates, such as connection failures and timeouts*
* *Bottlenecks are forming frequently. You can’t access the database, authentication fails, etc.*

***Tips for building highly scalable software –***

*Below are the eight tips that will help to build software that is easier to scale in the future.*

1. ***Opt for hosting in the cloud for better software scalability*** *- Host your applications, either in the cloud or on premises. Or you can use a hybrid approach. If you opt for the on-premises model, you will rely on your own infrastructure, accommodate your data storage, etc. This setup will limit your ability to scale and make it more expensive. Choosing cloud computing services will give you the possibility to access third-party resources instead of using your infrastructure. With the cloud, you have an almost unlimited possibility to scale up and down without having to invest in servers and other hardware.*
2. ***Use load balancing*** *- If you decide to scale horizontally, you will need to deploy load-balancing software to distribute incoming requests among all devices capable of handling them and make sure no server is overwhelmed. If one server goes down, a load balancer will redirect the server’s traffic to other online machines that can handle these requests.*
3. ***Cache as much as you can -*** *Cache is used to store static content and pre-calculated results that users can access without the need to go through calculations again. Cache as much data as you can to take the load off your database.*
4. ***Enable access through APIs*** *- End users will access your software through a variety of clients, and it will be more convenient to offer an application programming interface (API) that everyone can use to connect. An API is like an intermediary that allows two applications to talk. Make sure that you account for different client types, including smartphones, desktop apps, etc.*
5. ***Benefit from asynchronous processing*** *- This technique enables software scalability as it allows applications to run more threads, enabling nodes to be more scalable and handle more load. And if a time-consuming task comes in, it will not block the execution threat, and the application will still be able to handle other tasks simultaneously.*
6. ***Opt for database types that are easier to scale, when possible*** *- Some databases are easier to scale than others. For instance, NoSQL databases, such as MongoDB, are more scalable than SQL.*
7. *Choose microservices over monolith architecture, if applicable*
8. *Monitor performance to determine when to scale.*

*Challenges you might encounter while scaling –*

* *Accumulated technical debt.*
* *Scaling with Agile development methodology.*
* *Scalability testing.*
* *Scalability of third-party services.*
* *Understanding your application’s usage.*
* *Architectural restrictions*