Monolith and Microservices

***Monolith*** *- Monolithic architecture is the traditional approach to building software applications. In a monolith, all components of the application, including the user interface, business logic, and data storage, are tightly integrated into a single codebase and executed within a single runtime process.*

*Advantages of Monolithic Architecture:*

* *Simplicity: Monoliths are easier to develop, test, and deploy since all components are in one place.*
* *Performance: Monolithic applications often exhibit lower latency and faster communication between components due to in-process calls.*
* *Maintenance: Debugging and maintaining a monolith can be simpler because all the code is in one place.*
* *Versioning: Keeping components in sync is straightforward since they all share the same codebase.*

*4 reasons to stick up to a monolithic architecture –*

1. ***Your application is simple****. Small solutions which do not demand much business logic, superior scalability, and flexibility work better with a monolith.*
2. ***You need to launch it quickly****. In case you want to develop your application and use it as soon as possible, a monolithic model is the best choice. It works well when you aim to spend less initially and validate your business idea or want to keep your legacy system without further plans for modernization.*
3. ***You strive for lower software latency****. In monolith-based solutions, all communications are realized within a single instance of a deployed application. Since there are fewer network communications, it takes minimum time for a data packet to travel from one designated point to another.*
4. ***Your monolith is modular****. This architecture type stands for a code organization that is unified yet presupposes segmenting code into individual feature modules. If your code is well-organized and easy to observe within a monolith, there’s no need to switch to microservices. Dependent project*
5. ***Initial stage of Project****.*

***Monolithic application issues*** *–*

* *Large* *and Autonomous. Hence cannot change easily.*
* *Heavily interconnected and rely on each other, a tiny change needs to redeploy completely new version of application.*
* *Difficult to scale.* *An increase in the demand for the bandwidth of any one process would mean that the complete architecture needs to be scaled up.*
* *Monolithic applications might be susceptible to failure. This is because tightly coupled, essentially interdependent processes are easily affected if a single process goes down.*

***Microservice*** *- Microservice architecture, also known as ‘microservices,’ is defined as a development method that breaks down software into modules with specialized functions and detailed interfaces. It makes large systems easier to manage. It divides the systems into smaller units called services. Each service handles only one part of the software system. The services communicate with each other to make the system work as a whole.*

***Microservices –***

* *Microservices allow large applications to be split into smaller pieces that operate independently. Each ‘piece’ has its responsibilities and can carry them out regardless of what the other components are doing.*
* *As they run independently, the services can be deployed, updated, and scaled according to the demand for their specific functions.*
* *Isolation- Since each component of a microservices application operates independently, it's far easier to monitor the health and performance of each component and oversee the operation of the greater application. This isolation also makes it easier to identify and remediate faults such as restarting a failed module.*
* *Speed. Each microservices component is small, letting developers design, code, test and update a component in far less time than a traditional monolithic application design.*
* *Reusability. Modular application components are reusable by other applications, further easing application design investments and speeding development timeframes.*
* *Container compatibility. Microservices components are suited for deployment and management within virtual containers, using well-proven container and orchestration technologies such as Docker and Kubernetes.*
* *Resilience: Microservices are designed to be fault-tolerant (where fault tolerance is achieved through redundancy and automatic failover mechanisms), which means that if one service fails, it won't bring down the entire system.*
* *Technology Heterogeneity: Microservices architecture allows you to use different technologies for each service, so you can choose the best technology for the specific task or service. It can lead to better performance, scalability, and flexibility.*

***Disadvantages –***

* *independent microservices that communicate across a network, this significantly increases the application's architectural complexity.*
  + - *Scale 10 applications instead of one.*
    - *Secure 10 API endpoints instead of one.*
    - *Administer 10 Git repositories instead of one.*
    - *Build 10 packages instead of one.*
    - *Deploy 10 artifacts instead of one.*
  + *Communication between services is complex: Since everything is now an independent service, you have to carefully handle requests traveling between your modules.*
  + *More services equal more resources: Multiple databases and transaction management can be painful.*
  + *Global testing is difficult.*
  + *Debugging problems can be harder.*

***4 reasons to go for a microservices architecture***

1. *You need to develop a complex and scalable application. The microservices architecture will make scaling and adding new capabilities to your application much easier. So, if you plan to develop a large application with multiple modules and user journeys, a microservice pattern would be the best way to handle it.*
2. *You plan to release new features often. Using microservices allows you to considerably accelerate your time-to-market. Within this architecture type, engineering teams can build and deploy new functionalities separately in each service without a need to redesign the whole solution.*
3. *You aim at increasing fault tolerance. In microservice software, individual modules are isolated from one another. Therefore, if one system component fails, the other application parts won’t stop working properly.*
4. *You want to use several technologies within one software. The isolation of modules enables you to choose the most appropriate technology for each service—and they won’t contradict****.***

***Comparison –***

* *Monolithic architecture is a good choice when we must build a small-sized software for a standalone system. Microservice architecture is preferred when we have to develop large enterprise applications.*
* *Monolithic systems are easier to develop. They do not require any particular domain knowledge and expertise. Microservices are more challenging to develop.*
* *Maintenance and debugging – Easy in monolith and tedious in microservices*
* *Latency – good in monolith, not good in microservices*
* *Monolithic systems are easier to deploy. Microservices, the deployment is a more complex process.*
* *Scalability Issues - Microservices offers a scalable architecture. Monolithic systems, scaling them requires a lot of internal changes to the code. This could possibly disrupt the working of modules.*
* *To update a monolithic software, you need to take down the entire software. Updating microservices is somewhat simpler.*
* *Reusing any part of the code of a monolithic system is a very tedious process. Microservices architecture provides much more reusable components in the form of services.*
* *Polyglot(multiple tech) – Easier with microservice and tedious in monotith.*