Microservice Architecture

Context:

In the very beginning (60's) writing software was not just a piece of cake only Phd in computer science can perform that task. So as the demand of software development increases with the passage of time that becomes a huge problem to write a program as the software's were becoming large and complex. Scientist dealt with this problem with the **divide and conquer** rule.

So, in 70's scientists introduce the concepts like modularity, information hiding, separation of concern etc that lead to the **modular software development**. Large and complex software's were transforming into **loosely coupled and highly cohesive**. Loosely coupled means that every module should be independent or very less dependent on other modules so that if there is a change in one module that will not affect the other modules in the software. Highly cohesive means that every module should focus on one feature/function or similar feature or functions so that if that module had some type of bug or error that do not affect the whole program only that particular feature will be affected. In this modular software development modules communicate via **internal interfaces** (no physical network).

In 90's software's were becoming more complex and large although modularity is being used by the scientist to reduce complexity but as it's easy to cross the modular boundaries and misuse the application because of communication through internal interfaces, another software architecture came into scope which is called **Layered architecture**. In this architecture software application divided into different layers like presentation, business and database layers. In 97 Brian Foote and Joseph Yoder published the **Big ball of Mud** papers that highlighted the following issues:

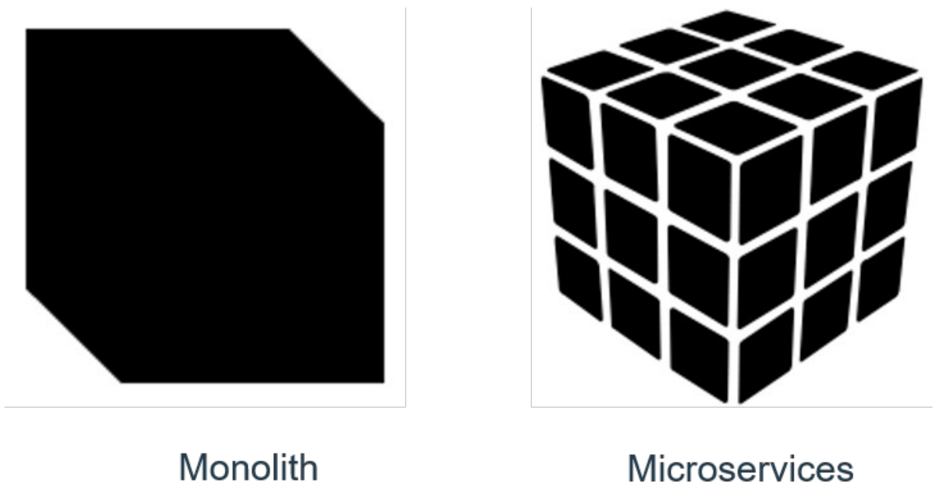
- Unregulated growth.
- Too many responsibilities.
- Lacks Proper Architecture.
- Speghetti Code.
- Seeping problems under the carpet.

In 2000's due to the rise in mobile internet and faster network software development pace was increased very rapidly. So engineers found that existing monolithic architecture approach cant handle this pace of development because in monolithic architecture there are a lot of limitations in Application scaling, Development Velocity, Development Scaling, Release Cycle, Modularization and Modernization companies like Facebook, Twitter, Uber etc came up with innovative ideas and aggressive approach leads to the growth of their applications.

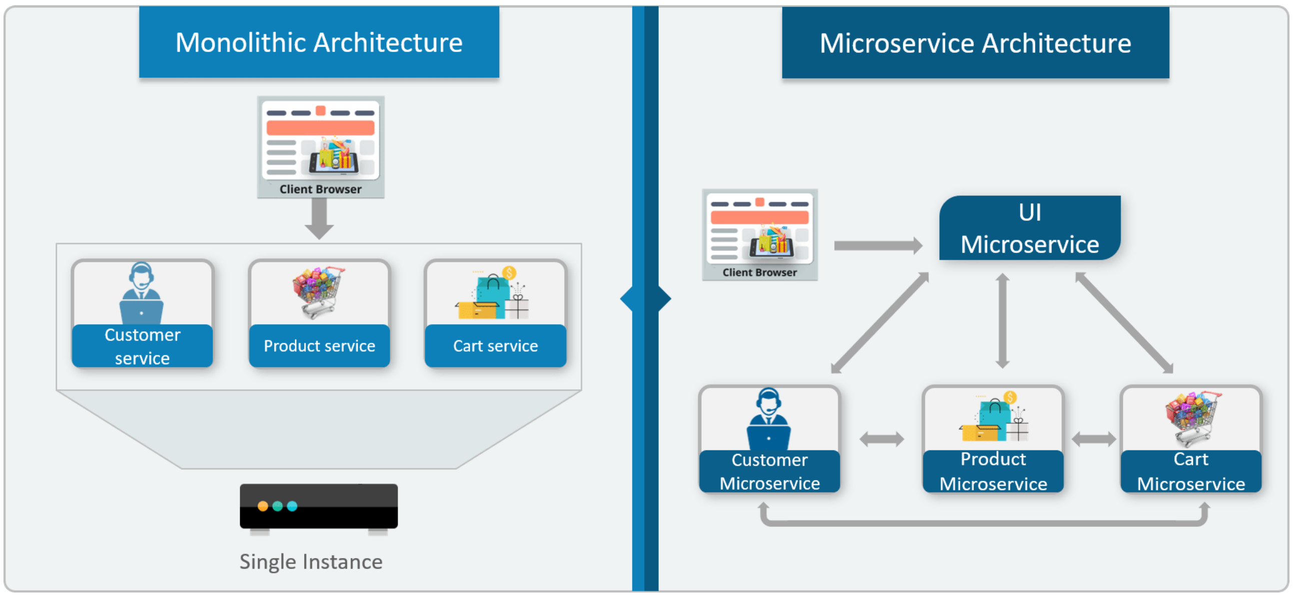
Introduction to MicroServices Architecture:

In 2010's there was a massive change in software industry with the arise of new **technologies** (i.e.cloud computing, containerization and DevOps) and **languages** (i.e. Golang, Rust, Swift, Python etc) and **software development models** (i.e. shift from waterfall to Agile) and **database technologies** (i.e. NoSQL, NewSQL etc.). To get benefit from all these modern technologies, languages and to handle these modern software development complexities microservice architecture was introduced in 2012.

> "In Microservice Architecture there are different modules each for each feature or functionality that are deployed autonomously without any particular language barrier (language agnostic way) and together communicating via external interface (Physical Network) they fulfill the business goals."



Comparison B/W Modular Monolithic and Microservice Architecture:

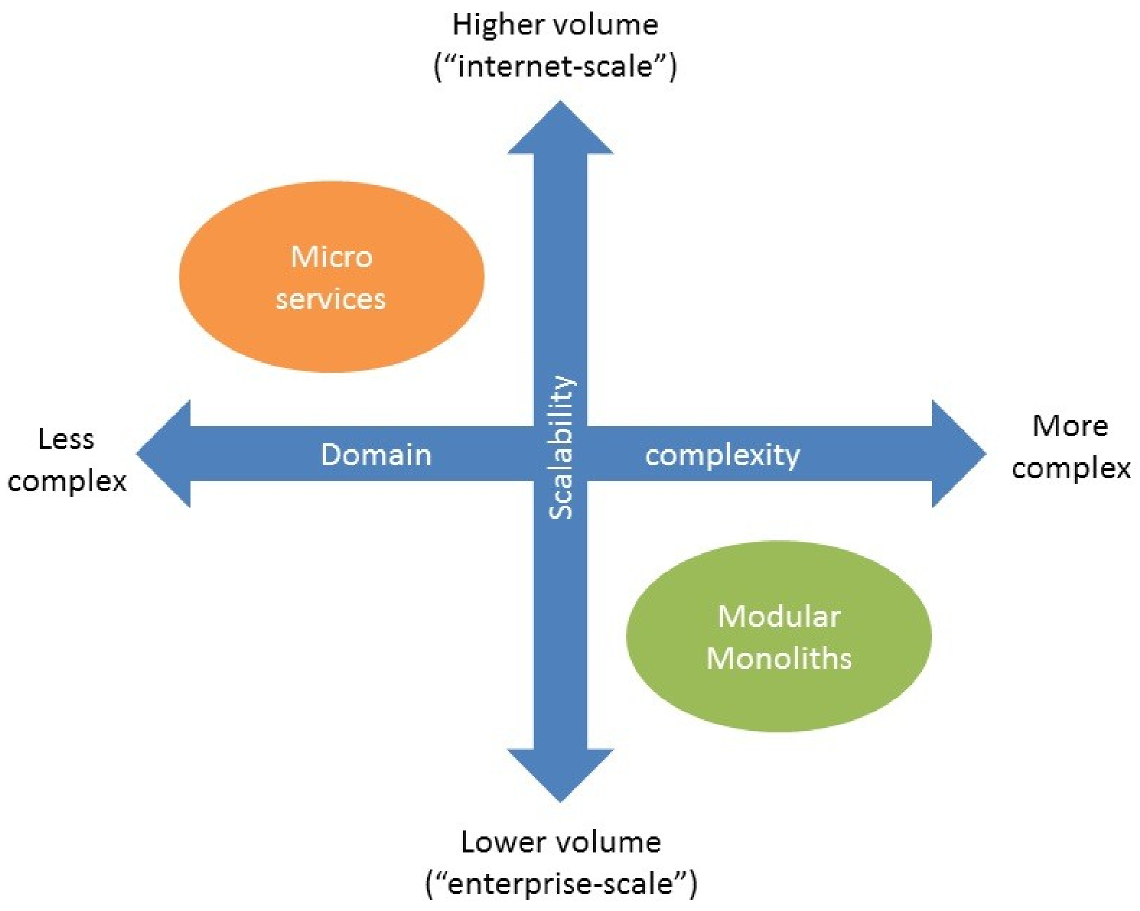


	Modular Monolithic Architecture	Microservice Architecture
Application Scaling	Monolithic Software development works as a single unit which means it cant support polyglot programming. So, horizontal scaling cant be implemented at its best.	Microservices works autonomously and different modules can be written using different languages and horizontal scaling can be implemented within seconds.
Design Complexity	It gives one solution for all type of business application. So in the beginning we just need to focus on logic of program	Every application has many different solution and choosing a best solution for a particular application is a must to do task.
Development Velocity	Introducing new features in a monolithic application is time consuming and very hectic due to its tightly coupled modules.	Its easy in Microservices to introduce new features because every module is independent and making changes in it do not affect the whole application.
Distributed system Complexities	System is less complex due to lesser hardware and operational complexities also lesser.	System is very complex due to lot of hardware involvements and operational complexities are also higher because taking care of one machine is easy as compared to more than one.
Development Scaling	Due to tightly coupled code new developer required a lot of time to write its first line and extra synchronization also required between developers. So increasing developer numbers cant assure that they will add more features in the applications.	Due to independent module system every coder can work freely on different features without interfering in others code and new developers also writes there first code very easily.
Operational Complexity	Complexity is in source code but easy to handle one system. So initial development velocity is high.	Complexity shifts from coding to operations due to complex systems. So initial development velocity is low.
Release Cycle	Size of monolithic code is directly proportional with the release cycle	Release Cycle is very small because it is not dependent on the source code.

	which means the more larger the code is the more larger will be the release cycle.	The size of code do not affect the release cycle because of independent deployment of modules.
Security	More the systems will be the more complex the security of them will be. In monolithic architecture number of systems is very small so the security is hi	There are a lot of systems involved in this architecture so the security became a very complex issue.
Modernization	Modernization of every successful application is mandatory because of many reasons like new technologies, methodologies etc. and modernization in monoliths is very expensive and time consuming because it is tightly coupled.	Modernization is easy because the application is loosely coupled and highly cohesive. New module can be written in newly released language.
Communication Complexity	Monolithic Architecture communicate via internal interface which means lower network latency.	Microservice Architecture communicates via external interface which means higher network latency.

Conclusion:

Microservice Architecture is not a solution for all the problems. Everything has its pros and cons and microservice has its own. We can say that for large organizations with huge coding work microservice architecture is the better solution to deal with the complexities with the selection of right solution for their application. While small or medium size organizations can slowly move towards microservices because taking aggressive approach towards microservices can lead to wrong selection of solution which will cost more in every aspect as compare to the monolithic architecture.


Higher volume
("internet-scale")
Micro services
Less complex
Domain
Scalability
complexity
More complex
Modular Monoliths
Lower volume
("enterprise-scale")