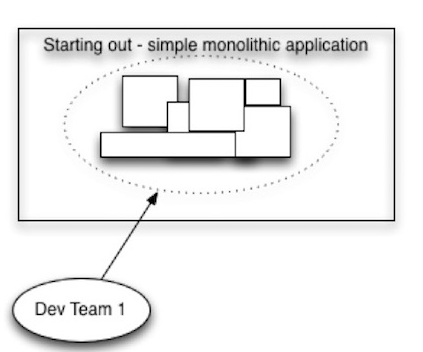
1. Background

Before jumping into the world of computer architectures and their features, advantages and drawbacks, we need to know why are we even discussing them? All software engineers and computer scientists need to use a set of rules, standards, and procedures to structure their code around, and the traditional way to do it was by using “Monolithic Architecture”.

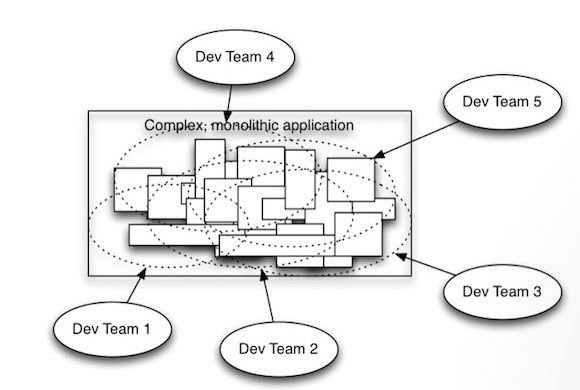
So, what are monoliths?

The below image describes a simple monolith architecture



However, the architecture cannot remain this simple for long. In the early days of software development, only those with PhD degrees in computers and mathematics had the right skillset to write computer code. But the development of BASIC in the 60s lowered the barrier, to the point that even non-doctoral students from different departments were able to write code. This led to a rapid expansion of computer applications, thus increasing the size and complexity of these software.

Now, the much bigger monolith looked like this:



Computer scientists tried to tackle this situation with the “Divide and Conquer” rule.

Now, what is that supposed to mean?

Different researches by different people led to the development of this process, some of these papers included:

* Introduction of the concepts of Modularity and Information hiding by David Parnas in his 1972 paper “T[he Criteria to be used in decomposing Systems into Modules](https://www.win.tue.nl/~wstomv/edu/2ip30/references/criteria_for_modularization.pdf)” by [*David Parnas*](https://en.wikipedia.org/wiki/David_Parnas)
* Introduction of the concept of Separation of Concern by Edsger W. Dijkstra in his paper “T[he role of scientific thought](http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html)”, 1974

These papers, alongwith research by others, led to the Modular Software Development Approach in the 70s, which can be described as “loosely coupled” (means the dependency between modules should be very low ) and “highly cohesive” (one module should focus on single or similar functionality).

But even the modular approach couldn’t work for long, as the soft Modular boundaries of software subsystems are easy to cross and misuse.

After analyzing different business applications, the “Big Ball of Mud” papers were published, which defines the term, according to which most business applications suffer from the following the problems:

* Unregulated growth
* Too many responsibilities
* Lacks proper Architecture
* Spaghetti Code
* Sweeping problems under the Carpet

With the internet coming to mobile devices and the rise of WiFi and smartphones, a Cambrian explosion happened in the software engineering world. Software began consuming the world, and various industrial segments like banking, hospitality, insurance etc began using software for their operations. Companies like **Facebook**, **Twitter**, **Uber**, **Netflix**, **Spotify** sprung up which increased the dependency of consumers on software products. With such fast growth and steadily increasing numbers of software consumers, it was deduced that the traditional method aka “Monolithic Architecture” is inept at handling such large scale and fast-paced applications.

1. What are microservices and why are they better?

Microservice architecture, or simply, microservices, is an architectural paradigm that organizes an application as a collection of services that are

* Loosely coupled
* Highly cohesive
* Organized around business capabilities
* Independently deployable
* Owned by a small team

Microservices ensure that large, complex applications are able to evolve rapidly, frequently and reliably. Also, evolving the technology-stack becomes easier for organizations.

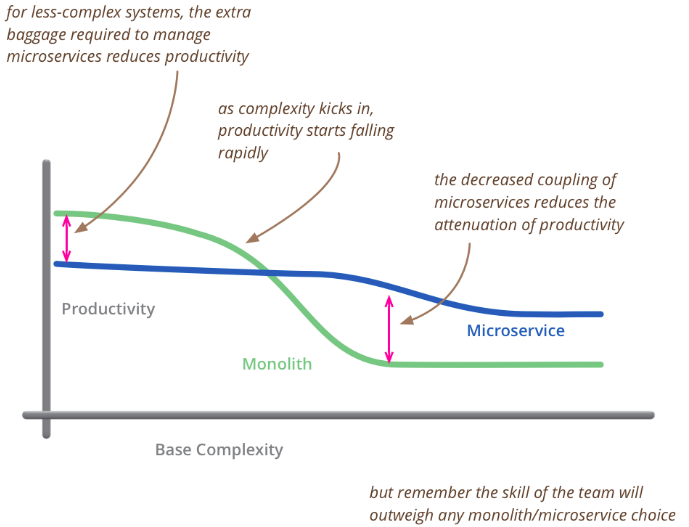
When making a comparison between monolithic architecture and microservices, microservices comes out as the clear winner. Let’s see how:

1. It is difficult to achieve horizontal scaling with monolith architecture, however with microservices such scaling can be performed within seconds.
2. It takes greater time for the application to be developed as a developer needs to change different parts of the architecture to add a new feature due to the tightly coupled modules of monolith architecture, however in microservices the modules are loosely coupled and small in size, thus making time to market shorter than monolith
3. Working on a gigantic code base of a monolithic architecture is comparatively difficult for developers. The high cognitive load makes it frustrating for developers to write productive code. However in microservices, individual developers are responsible for individual modules which are very lightly, somewhat non-existingly, dependent on each other.
4. It takes months, and even years, for monoliths to be released into the market. Such high release cycles put companies under competitive disadvantages. Such is not the case with microservices, as the development of a single module doesn’t cause any strain on other modules.
5. Even if we try modularization in monoliths, their boundaries start to merge together and then a time comes when the modules become tightly coupled instead of being loosely coupled. However, in microservices, the boundaries are physical i.e they are external interfaces which makes it hard for the modules to merge together even when the application becomes large.
6. With the changing times, there are many things which need to be upgraded or modernized for the company to be able to have an edge over the market. Modernization is somewhat expensive in monoliths, but is very easy for developers in microservices since each module is independently deployable. As a result, the Software Release Cycle in Microservices is much smaller.

The microservices approach also complements DevOps, which helps us accelerate the delivery of software from development into production by eliminating the barriers between software development and IT operations. In a DevOps environment, developers and operations engineers work together as a team to supply higher-quality software faster.

This helps us create rapid changes to applications within very short intervals of production schedules. By simultaneously utilizing microservices and DevOps, software organizations can accomplish things that were considered impossible not long ago. Amazon for example is in a position to release new code every 11.7 seconds, while Netflix, per day, deploys thousands of lines of new code,

1. Disadvantages of microservices
   1. Monolith architectures, with all their disadvantages, have one advantage over microservices, they provide a “one size fits all” strategy for business developers. However, in microservices, each module needs to be independently designed and cannot be replicated for other applications.
   2. If monoliths were complex in terms of code, microservices are complex in terms of operations like keeping track of logging and the monitoring of several modules and the existing logging/monitoring tools became less suited to the more complex needs of microservices. Tracing is also very important in Microservices to measure the performance/latency of individual Microservices for a Service Request. As renowned Computer Scientist and Microservice Guru [***Martin Fowler***](https://martinfowler.com/) has pointed out, the initial Development Velocity of Microservice Architecture is lower compared to Monolithic Architecture due to the Operational Complexities.



* 1. Security is a challenge since in microservices we have to ensure that the many different systems are secure, unlike monoliths were we had to secure only one system.,
  2. It isn’t possible for microservices to have their own database, since Microservices need to share data between themselves to fulfill the Business goal. Consistency in the distributed databases is not recommended for two reasons:
     1. It does not Scale and many Modern databases do not support it
     2. Most of the modern NoSQL Databases only offer Eventual Consistency which needs careful design
  3. Due to the physical network communication between microservices, the increased network latency brings about communication complexities. There are two ways where microservices can communicate with each other,
     1. Synchronous Communication - it is easier to implement using REST and gRPC but leads to a distributed monolith
     2. Asynchronous Communication - is relatively complex to implement using Message Queue and Message Broker. It is more flexible than synchronous.

1. The correct usage of Microservices, is the key.

It is true that designing and implementing Microservice Architecture is challenging and needs a Paradigm Shift compared to Monolithic Software, Used effectively, microservice architectures allow you to scale your application as the number of developers working on your application increases. The key is to build applications without creating a complex, unwieldy beast at the macro level. That means keep tracking each time a new service is added to your system or a new connection between microservices is made.

Microservices are not exactly a panacea, the benefits of microservices make it a win-win for increasing numbers of modern software organizations. For large Enterprises which normally develop complex Softwares, Microservice Architecture is the only way to tackle complexity and to be competitive in the market. For small to medium enterprises also, Microservice Architecture should be used for sustainable software development which can give long term benefits.

[1] Brian Foote and Joseph Yoder, *Big Ball of Mud.* Fourth Conference on Patterns Languages of Programs (PLoP '97/EuroPLoP '97) Monticello, Illinois, September 1997

[2] <https://microservices.io/>