The purpose of this article is to be a deep understanding of architectural models for building a software application. It also goes through comparison of two best models and analyzing them from the different context of view.

What is a monolithic application?

A monolithic application is built as a single, self-contained unit. Usually, such a solution comprises a client-side user interface, a server side-application, and a database. It is unified and all the functions are managed and served in one place.

Normally, monolithic applications have one large code base and lack modularity. All code exists inside a single codebase, which means that the entire application is deployed at the same time, and scaling is achieved by adding additional nodes.

What is microservices-based application?

Microservices are an architectural style where an application is broken into a series of modules, each associated with a specific business objective.

So, all the services have their own logic and the database as well as perform the specific functions. Within a microservices architecture, the entire functionality is split up into independently deployable modules which communicate with each other through defined methods called APIs (Application Programming Interfaces). Each service covers its own scope and can be updated, deployed, and scaled independently.

Comparative analysis

Thanks to rousing endorsements from the likes of Google, Amazon, and Netflix, microservices are continuing to rise in popularity.

The monolithic architecture is a traditional way of building applications, but no longer are software monoliths able to meet the demands of today’ s fast-paced competitive business landscape. Microservices offer agile, customizable software that allows for fast deployments, frequent updates, and increased scalability.

MONOLITHIC ARCHITECTURE ADVANTAGES

- **Easier debugging and testing.** In contrast to the microservices architecture, monolithic applications are much easier to debug and test. Since a monolithic app is a single indivisible unit, you can run end-to-end testing much faster.
- **Simple to deploy.** Another advantage associated with the simplicity of monolithic apps is easier deployment. When it comes to monolithic applications, you do not have to handle many deployments – just one file or directory.
- **Simple to develop.** As long as the monolithic approach is a standard way of building applications, any engineering team has the right knowledge and capabilities to develop a monolithic application.
- **Less cross-cutting concerns.** Cross-cutting concerns are the concerns that affect the whole application such as logging, handling, caching, and performance monitoring. In a monolithic application, this area of functionality concerns only one application so it is easier to handle it.

The monolithic approach also comes with performance advantages, since shared-memory access is faster than inter-process communication (ICP) for example.

MONOLITHIC ARCHITECTURE DISADVANTAGES

- **Understanding.** When a monolithic application scales up, it becomes too complicated to understand. Also, a complex system of code within one application is hard to manage.
- **Making changes**. It is harder to implement changes in such a large and complex application with highly tight coupling. Any code change affects the whole system, so it has to be thoroughly coordinated. This makes the overall development process much longer.
- **Scalability**. You cannot scale components independently, only the whole application.
- **New technology barriers**. It is extremely problematic to apply a new technology in a monolithic application because then the entire application must be rewritten.

Quite understandably, companies facing the need to move faster, and drive innovation are trying to find ways around these restrictions.

MICROSERVICES ARCHITECTURE ADVANTAGES

- **Independent components.** Firstly, all the services can be deployed and updated independently, which gives more flexibility. Secondly, a bug in one microservice has an impact only on a particular service and does not influence the entire application. Also, it is much easier to add new features to a microservice application than a monolithic one.
- **Easier understanding.** Split up into smaller and simpler components, a microservice application is easier to understand and manage. You just concentrate on a specific service that is related to a business goal you have.
- **Better scalability.** Another advantage of the microservices approach is that each element can be scaled independently. So, the entire process is more cost- and time-effective than with monoliths when the whole application must be scaled even if there is no need in it. In addition, every monolith has limits in terms of scalability, so the more users you acquire, the more problems you have with your monolith. Therefore, many companies, end up rebuilding their monolithic architectures.
- **Flexibility in choosing the technology.** The engineering teams are not limited by the technology chosen from the start. They are free to apply various technologies and frameworks for each microservice.
- **The higher level of agility.** Any fault in a microservices application affects only a particular service and not the whole solution. So, all the changes and experiments are implemented with lower risks and fewer errors.

MICROSERVICES ARCHITECTURE DISADVANTAGES

- **Extra complexity.** Since a microservices architecture is a distributed system, you must choose and set up the connections between all the modules and databases. Also, if such an application includes independent services, all of them must be deployed independently.
- **System distribution.** A microservices architecture is a complex system of multiple modules and databases so all the connections must be handled carefully.
- **Cross-cutting concerns.** When creating a microservices application, you will have to deal with a number of cross-cutting concerns. They include externalized configuration, logging, metrics, health checks, and others.
- **Testing.** A multitude of independently deployable components makes testing a microservices-based solution much harder.

Why would one want to migrate from one type of application to another?

Recently, many companies have moved from their monolithic architecture to microservices architecture. Moreover, Enterprise applications are usually born as monoliths. The real reason to build a microservices-based application is that it will enable the teams in your organization to work more independently. And as part of working more independently, they will also work more efficiently.

Microservices take precedence, however, once a system grows too large and complex. A common practice is to break the monolith into small autonomous pieces, each deployed and sustained by an agile team while leaving the team collaboration issues behind.

When a business application, and a development team, get bigger and reach a certain size, companies face a critical management and cooperation bottleneck. Moreover, if a software product is based on a massive monolith architecture, technological challenges are also faced. In such cases, businesses require a solution to fix the workflow and enhance collaboration on the project. A microservice architecture is an answer to the problems associated with the traditional back-end monolith once its complexity calls for higher scalability.

When starting a new project, which approach is best?

If your business idea is fresh and you want to validate it, you should start with a monolith. With a small engineering team aiming to develop a simple and lightweight application, there is no need to implement microservices. This way, a monolithic application will be much easier to build, make changes, deploy, and provide testing.

Typical scenarios for starting with a monolith are:

1. **Your team is small and at a founding stage**. A starting team of about two to five members will find a monolith architecture ideal to focus on.
2. **You are building a proof of concept.** New products will pivot and evolve as you figure out what will be useful to your users. For this, a monolith is perfect as it allows for rapid product iteration.
3. **Foreseeable scale and complexity.** When your application does not require advanced scalability and the complexity is manageable, then a monolith architecture is the best road to go down.

The microservices architecture is more beneficial for complex and evolving applications. It offers effective solutions for handling a complicated system of different functions and services within one application. Microservices are ideal when it comes to the platforms covering many user journeys and workflows. But without proper microservices expertise, applying this model would be impossible.

When To Start with Microservices

Here are some scenarios that indicate that you should start your next project using microservices:

- **You Require Quick, Independent Service Delivery:** If it is snappy, isolated service delivery that you need, microservices are your best bet. However, depending on the size of your team, it can take some time before you see any service delivery gains versus starting with monolith.
- **A Piece of Your Platform Needs to Be Extremely Efficient:** If your business is doing intensive processing of petabytes of log volume, you' ll likely want to build that service out in a very efficient language (i.e., C++) while your user dashboard may be built in Ruby on Rails.
- **You Plan to Scale Your Team:** Starting with microservices gets your team used to developing in separate small teams from the beginning, and having teams separated by service boundaries makes scaling your development organization easier.

We do not attempt to take on microservices just because other engineering teams are having success with them. It depends on our own context, evaluated against the above considerations, is the key to deciding if new project should start with monolith or microservices.

How do these approaches compare in the context of maintainability?

"It Depends" . Just like most things in software, context is the key. An approach might work beautifully for a team and might not for the other.

When you need to edit, update, or add functionalities, the code base increases making the application grow. When an application becomes too big and complex, it is also **difficult to entirely understand it** and make changes, even small ones, without touching the entire system.

Microservices architecture, in my opinion, is breaking one or more monoliths into multiple smaller systems. These are more maintainable, independently developed and deployed pieces of software based on business functions. It becomes very decoupled. The blast radius of each change is controlled. That is the reason for rapid microservice adoption.

In a monolithic application, having a large code base is not the only challenge that you'll face. Having a large team to handle such a code base is one more problem that will affect the growth of the business and application.

In conclusion, evolving monolithic applications and corporate legacy systems provides the company with many advantages that also means **a reduction in time-to-market**, **reduction of IT costs**, and **customer experience improvement**.