This paper highlights two most well known architectural models of building a software application. It also goes through the advantages and disadvantages of each model helping to better understand their difference and mission.

What is a monolithic application?

A monolithic application is built as a single 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.

What is microservices-based application?

Microservices (or microservices architecture) are a cloud native architectural approach in which a single application is composed of many loosely coupled and independently deployable smaller components, or services. These services typically have their own technology stack, inclusive of the database and data management model; communicate with one another over a combination of REST APIs, event streaming, and message brokers; and are organized by business capability, with the line separating services often referred to as a bounded context.

Comparative analysis of these two architectural approaches

The monolithic architecture is considered to be a traditional way of building applications. A monolithic application is built as a single and indivisible 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. If developers want to update or change something, they access the same code base. So, they make changes in the whole stack at once.

Strengths of the Monolithic Architecture

- Less cross-cutting concerns.
- Easier debugging and testing.
- Simple to deploy.
- Simple to develop.

Weaknesses of the Monolithic Architecture

- When a monolithic application scales up, it becomes too complicated to understand.
- It is harder to implement changes in such a large and complex application with highly tight coupling.
- You cannot scale components independently, only the whole application.
- It is extremely problematic to apply a new technology in a monolithic application because then the entire application has to be rewritten.

While a monolithic application is a single unified unit, a microservices architecture breaks it down into a collection of smaller independent units. These units carry out every application process as a separate service. 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.

Strengths of the Microservice Architecture

- Independent components.
- Easier understanding.
- Better scalability.
- Flexibility in choosing the technology
- The higher level of agility.

Weaknesses of the Microservice Architecture

- Extra complexity.
- System distribution health handling.
- Cross-cutting concerns, such as externalized configuration, logging, metrics, health checks, and others.
- 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?

Enterprise applications are usually born as monoliths. This practice seems reasonable since the support is less demanding and the system works well from a limited scopes. 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.

Enterprises usually embrace this kind of solution only when the practical need occurs. In most cases, a development team of more than 20-25 members starts suffering from collaboration difficulties while maintaining the monolith applications. So, the initiative to adopt microservices often comes from the development team or a software vendor.

When starting a new project, which approach is best?Why?

Depending on the requirements of a company or customer, the development team should decide on this. For different needs there comes a different approach.
When to go with a monolithic architecture?
In some cases, a monolithic approach is a time-tested strategy:

- you plan to build a small app.
- you don't plan to grow your team. In this case, designing and managing a complex system is not the best way to go.
- you're at the ideation stage. If you are at the first stage of SDLC, your product is likely to grow over time. A monolithic architecture allows for fast iterating.
- you are building an MVP. At this stage, your goal is to gather feedback from first users as soon as possible and monolith apps are the quickest way.

When to go with microservices?
Plenty of companies switched to the microservice architecture after their customer demand increased significantly. Among them are Amazon, PayPal, Spotify, and many more.

- you want to build a large-scale solution.
- you have a lot of time on your hands since microservice architecture requires thorough planning.
- scalability is critical for your project and you plan to grow your development team.
- you need to use different languages for different elements like C++ for backend and Rails for the user interface.
- you want to build multiple independent teams that would work on different functions of your solution.

How do these approaches compare in the context of maintainability?

Coming to the maintenance question, it seems that microservices are winning the race because they tackle the problem of complexity by decomposing applications into a set of manageable services which are much faster to develop, and much easier to understand and maintain.

On the other hand, monolith is easily maintained being a lightweight application, which, in most cases, does not suit large companies having a lot of features and functionalities to be implemented. This leads to a huge amount of code that requires decoupling, otherwise it would be harder to be maintained.