Saad Arshed

38 Followers · About Follow

Monolithic vs SOA vs Microservices — How to Choose Your Application Architecture

Saad Arshed Nov 29, 2018 · 4 min read

Photo by rawpixel on Unsplash

Why does architecture matter? You want to build a product can scale, distribute, and helps you with the speed to market. Your development, testing, and production processes have to be Agile. You need to bring your idea to market fast. Then you have limited human capital resources. All this brings in complexity and pressure for architects and founders. Your choice can make or break your venture.

In this article, I will talk about the options that you have. I will also try to help you understand the prime differences and pro and cons of each approach. So, let's look into the options that you have — **Monolithic Architecture, Service-Oriented Architecture, or the latest MicroServices Architecture**.

Monolithic Architecture

Monolithic Architectures are synonymous with **n-Tier applications**. Here you are trying to separate concerns and decompose your code base into functional components. In essence, you are building a single web artifact and then you trying to decompose the application into layers.

In most of the cases, you will have **Presentation Layer**, then the **Business Logic Layer**, and finally the **Data Access Layer**. The idea behind this segregation is to work with any component of the architecture independent of the one underneath or above.

Cons:

This architecture resulted in massive coupling issues. Every time you have to build, test, or deploy, you are essentially playing with the entire code base. And it would take days for any function to go live. A single deployment might contain data access components, business logic components, web services and so on.

The agility of this architecture was a massive pain.

The infrastructure costs associated were also a major consideration. In case a single component is under load and requires scaling, you will have to add resources for the entire application. This means a bad performing part of your software architecture can bring the entire structure down or you pay a truckload of money to keep it up and running.

SOA Architecture

The natural transition from the monolithic application was to use service-based architecture (SOA). Using this approach you would decouple your application in smaller modules. All the services would then work with an aggregation layer that can be termed as a bus.

This architecture had a good way of decoupling and communication. Each service can communicate using a standard business processing layer. This layer separates the internal and external elements of the system

Cons:

This aggregation layer (SOA Bus) became the biggest challenge to handle. The issue was the addition of the operational logic to the bus. As this layer got bigger and bigger with more and more components added to the system, so came the issues of system coupling.

As per experts, another biggest issue came in the form of error handling. With this architecture, you would either get a 200 or 500 response and nothing in between.

The bloating of aggregation made this architecture fall out of architects' grace. People started looking towards monolithic architectures or moved towards the Micro-Services Architecture.

Micro-Services Architecture

Essentially micro-services were the evolution to the limitation of the SOA architecture. This architecture enabled the decoupling or decomposition of the system into discrete work units.

The art and skill of getting this architecture right come from the ability to define a micro-service. Too granular or too broad can destroy your architecture. You can use business cases, hierarchical, or domain separation to define each micro-service.

This architecture enabled a **true polyglot deployment**, where you can use different languages or frameworks to work together. So, the front end developers might be working with React, while the BackDeveloper are using C#, and the data team is working with Python. All the services can talk to one another and utilize the resources as needed.

All the communication between the services in over **ReST over HTTP**. One of the best ways to configure all the service communication is via an **API Proxy**.

This architecture also renders itself well suited for the **cloud-native deployment**. Done right, this architecture does help you in building cloud-native services. Most of the architects work with micro-services to move to a cloud-native platform.

Remember a monolithic application can be deployed on cloud and you can still enjoy the benefits of scaling of compute and storage resources.

Cons:

Now, this ability to call any services adds a lot of flexibility to the architecture. But it does come at a cost of payloads and latency. With the addition of more micro-services, network communication can increase dramatically. This can lead to latency and in case of an increase in call volumes, could lead to congestion and latency across the whole network.

A single blocking call can impact other services. So, this architecture led to the concept of **distribution tax**.

Summing Up:

In my opinion, if you are trying to build a new product with limited resources and programming talent, building a monolithic solution might not be a bad option. It gives you the ability to get to market early and help you build your MVP and test the product-market fit. Once the validation is in, you might want to opt for the MicroService architecture.

Remember, each architecture has had its utility during its time and might still serve a need. It all depends upon the project scope and the stage you are in.

Thank you for reading! If you find this helpful go ahead and give it a few👏 so that others can find it.

App2Dev.com

Originally Published on App2Dev.com/Blog

Microservices Monolithic Application Architecture Application Development Startup