Best Architecture for an MVP: Monolith, SOA, Microservices, or Serverless?

89555 VIEWS · 14 MIN · APR 18, 2019

Anastasia D.
Copywriter

TAGS: Web MVP

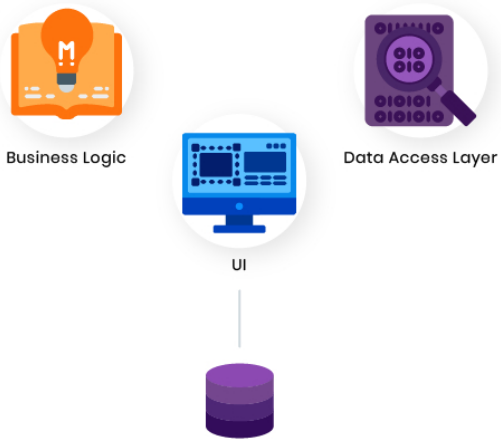
SHARE

Creating a new product is all about risk. And choosing the right architecture is an essential step toward success. If you're considering between a monolithic, service-oriented, microservice, and serverless architecture, this blog post will help you make the right choice.

CONTENTS

Monolithic architecture

Monolith is an ancient word referring to a huge single block of stone. Though this term is used broadly today, the image remains the same across fields. In software engineering, a monolithic pattern refers to a single indivisible unit. The concept of monolithic software lies in different components of an application being combined into a single program on a single platform. Usually, a monolithic app consists of a database, client-side user interface, and server-side application. All the software's parts are unified and all its functions are managed in one place. Let's look at the structure of the monolithic software in detail.



Monolithic Architecture

A monolithic architecture is comfortable for small teams to work with, which is why many

startups choose this approach when building an app. Components of monolithic software are interconnected and interdependent, which helps the software be self-contained. This architecture is a traditional solution for building applications, but some developers find it outdated. However, we believe that a monolithic architecture is a perfect solution in some circumstances.

> Even though we had had these positive experiences of using microservices at Google, we [at Scaylr] went for a monolith route because having one monolithic server means less work for us as two engineers.
>
> **Steven Czerwinski, Head of Engineering at Scaylr**

In order to find out whether this solution is good for your business, let's consider its pros and cons.

Pros of a monolithic architecture

Simpler development and deployment

There are lots of tools you can integrate to facilitate development. In addition, all actions are performed with one directory, which provides for easier deployment. With a monolithic core, developers don't need to deploy changes or updates separately, as they can do it at once and save lots of time.

Fewer cross-cutting concerns

Most applications are reliant on a great deal of cross-cutting concerns, such as audit trails, logging, rate limiting, etc. Monolithic apps incorporate these concerns much easier due to their single code base. It's easier to hook up components to these concerns when everything runs in the same app.

Better performance

If built properly, monolithic apps are usually more performant than microservice-based apps. An app with a microservices architecture might need to make 40 API calls to 40 different microservices to load each screen, for example, which obviously results in slower performance. Monolithic apps, in turn, allow faster communication between software components due to shared code and memory.

Cons of a monolithic architecture

Codebase gets cumbersome over time

In the course of time, most products develop and increase in scope, and their structure becomes blurred. The code base starts to look really massive and becomes difficult to understand and modify, especially for new developers. It also gets harder to find side effects and dependencies. With a growing code base quality declines and the integrated development environment (IDE) gets overloaded.

development environment (IDE) gets overloaded.

Difficult to adopt new technologies

If there's a need to add some new technology to your app, developers may face barriers to adoption. Adding new technology means rewriting the whole application, which is costly and time-consuming.

Limited agility

In monolithic apps, every small update requires a full redeployment. Thus, all developers have to wait until it's done. When several teams are working on the same project, agility can be reduced greatly.

The bottom line

The monolithic model isn't outdated, and it still works great in some cases. Some giant companies like Etsy stay monolithic despite today's popularity of microservices. Monolithic software architecture can be beneficial if your team is at the founding stage, you're building an unproven product, and you have no experience with microservices. Monolithic is perfect for startups that need to get a product up and running as soon as possible. However, certain issues mentioned above come with the monolithic package.

SOA

A service-oriented architecture (SOA) is a software architecture style that refers to an application composed of discrete and loosely coupled software agents that perform a required function. SOA has two main roles: a service provider and a service consumer. Both of these roles can be played by a software agent. The concept of SOA lies in the following: an application can be designed and built in a way that its modules are integrated seamlessly and can be easily reused.

Pros of SOA

Reusability of services

Due to the self-contained and loosely coupled nature of functional components in service-oriented applications, these components can be reused in multiple applications without influencing other services.

Better maintainability

Since each software service is an independent unit, it's easy to update and maintain it without hurting other services. For example, large enterprise apps can be managed easier when broken into services.

Higher reliability

Services are easier to debug and test than are huge chunks of code like in the monolithic

approach. This, in turn, makes SOA-based products more reliable.

Parallel development

As a service-oriented architecture consists of layers, it advocates parallelism in the development process. Independent services can be developed in parallel and completed at the same time. Below, you can see how SOA app development is executed by several developers in parallel:

Cons of SOA

Complex management

The main drawback of a service-oriented architecture is its complexity. Each service has to ensure that messages are delivered in time. The number of these messages can be over a million at a time, making it a big challenge to manage all services.

High investment costs

SOA development requires a great upfront investment of human resources, technology, and development.

Extra overload

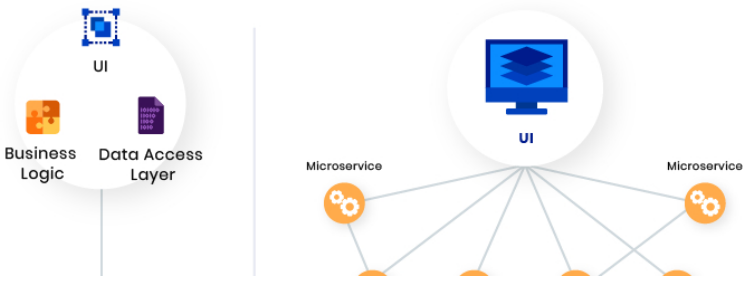
In SOA, all inputs are validated before one service interacts with another service. When using multiple services, this increases response time and decreases overall performance.

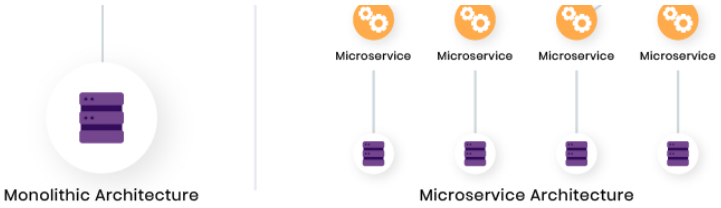
The bottom line

The SOA approach is best suited for complex enterprise systems such as those for banks. A banking system is extremely hard to break into microservices. But a monolithic approach also isn't good for a banking system as one part could hurt the whole app. The best solution is to use the SOA approach and organize complex apps into isolated independent services.

Microservice architecture

Microservice is a type of service-oriented software architecture that focuses on building a series of autonomous components that make up an app. Unlike monolithic apps built as a single indivisible unit, microservice apps consist of multiple independent components that are glued together with APIs.





The structure of the microservices and monolithic architecture in comparison

The microservices approach focuses mainly on business priorities and capabilities, whereas the monolithic approach is organized around technology layers, UIs, and databases. The microservices approach has become a trend in recent years as more and more enterprises become agile and move toward DevOps.

> Microservices are important simply because they add unique value in a way of simplification of complexity in systems. By breaking apart your system or application into many smaller parts, you show ways of reducing duplication, increasing cohesion and lowering your coupling between parts, thus making your overall system parts easier to understand, more scalable, and easier to change.
>
> **Lucas Kraus, author of Microservices**

There are lots of examples of companies that have evolved from a monolithic approach to microservices. Among the most prominent are Netflix, Amazon, Twitter, eBay, and PayPal. In order to determine whether microservices are suitable for your project, let's define the pros and cons of this approach.

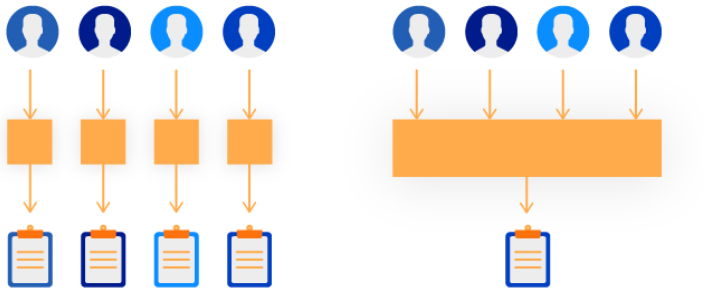
Pros of microservices

Easy to develop, test, and deploy

The biggest advantage of microservices over other architectures is that small single services can be built, tested, and deployed independently. Since a deployment unit is small, it facilitates and speeds up development and release. Besides, the release of one unit isn't limited by the release of another unit that isn't finished. And the last plus here is that the risks of deployment are reduced as developers deploy parts of the software, not the whole app.

Increased agility

With microservices, several teams can work on their services independently and quickly. Each individual part of an application can be built independently due to the decoupling of microservice components. For example, you may have a team of 100 people working on the whole app (like in the monolithic approach), or you can have 10 teams of 10 people developing different services for the app. Let's imagine this visually.



Increased agility allows developers to update system components without bringing down the application. Moreover, agility provides a safer deployment process and improved uptime. New features can be added as needed without waiting for the entire app to launch.

Ability to scale horizontally

Vertical scaling (running the same software but on bigger machines) can be limited by the capacity of each service. But horizontal scaling (creating more services in the same pool) isn't limited and can run dynamically with microservices. Furthermore, horizontal scaling can be completely automated.

Cons of microservices

Complexity

The biggest disadvantage of microservices lies in their complexity. Splitting an application into independent microservices entails more artifacts to manage. This type of architecture requires careful planning, enormous effort, team resources, and skills. The reasons for high complexity are the following:

- Increased demand for automation, as every service should be tested and monitored
- Available tools don't work with service dependencies
- Data consistency and transaction management becomes harder as each service has a database

Security concerns

In a microservices application, each functionality that communicates externally via an API increases the chance of attacks. These attacks can happen only if proper security measurements aren't implemented when building an app.

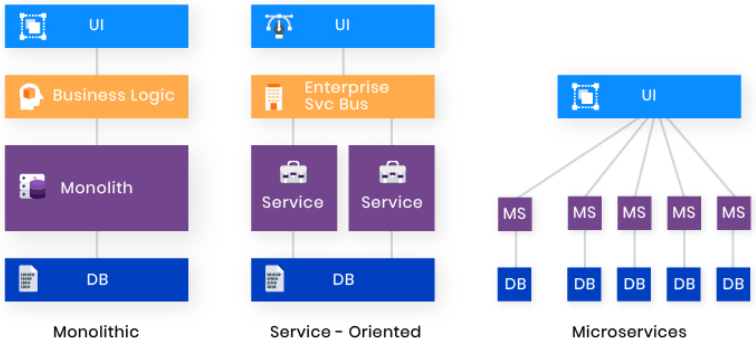
Different programming languages

The ability to choose different programming languages is two sides of the same coin. Using different languages make deployment more difficult. In addition, it's harder to switch programmers between development phases when each service is written in a different language.

The bottom line

Microservices are good, but not for all types of apps. This pattern works great for evolving applications and complex systems. Consider choosing a microservices architecture when you have multiple experienced teams and when the app is complex enough to break it into services. When an application is large and needs to be flexible and scalable, microservices are beneficial.

Now we can compare these three software architectures to define the differences between them visually.



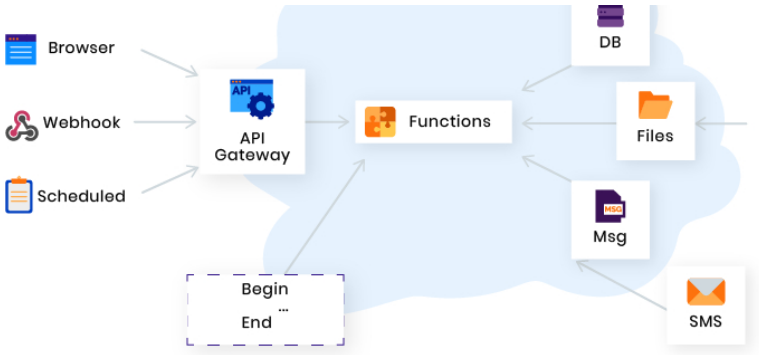
Monolithic apps consist of interdependent, indivisible units and feature very low development speed. SOA is broken into smaller, moderately coupled services, and features slow development. Microservices are very small, loosely coupled independent services and feature rapid continuous development.

Serverless architecture

Serverless architecture is a cloud computing approach to building and running apps and services without the need for infrastructure management. In serverless apps, code execution is managed by a server, allowing developers to deploy code without worrying about server maintenance and provision. In fact, serverless doesn't mean "no server." The application is still running on servers, but a third-party cloud service like AWS takes full responsibility for these servers. A serverless architecture eliminates the need for extra resources, application scaling, server maintenance, and database and storage systems.

The serverless architecture incorporates two concepts:

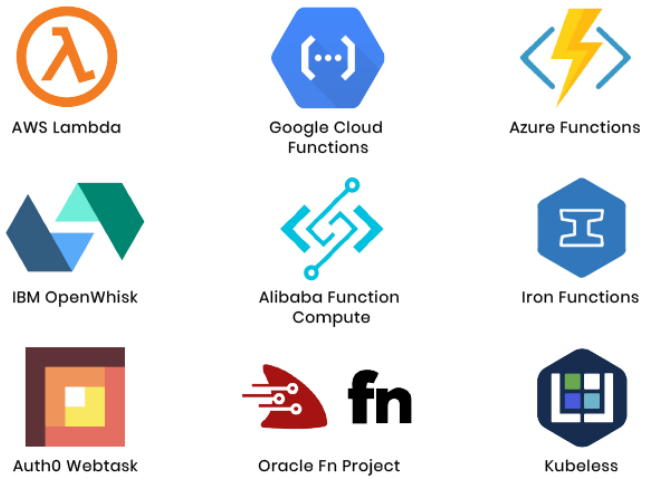
- **FaaS (Function as a Service)** – a cloud computing model which allows developers to upload pieces of functionality to the cloud and let these pieces be executed independently
- **BaaS (Backend as a Service)** – a cloud computing model which allows developers to outsource backend aspects (database management, cloud storage, hosting, user authentication, etc.) and write and maintain only the frontend part



This is how a serverless structure looks schematically

When using a serverless architecture, developers can focus on the product itself without worrying about server management or execution environments. This allows developers to focus on developing products with high reliability and scalability.

There are a lot of cloud vendors on the market. Here are some of the top serverless computing providers:



Top serverless computing providers

To know if this architecture type is what your project needs, let's define the benefits and drawbacks of implementing a serverless model.

Pros of a serverless architecture

Easy to deploy

In serverless apps, developers don't need to worry about infrastructure. This allows them to focus on the code itself. Serverless architecture allows you to spin up an app extremely fast, as deployment takes only hours or days (compared to days or weeks with a traditional approach).

Lower costs

Going serverless reduces costs. Since you don't need to handle databases, some logic, and

Using serverless reduces costs. Since you don't need to handle databases, some logic, and servers, you can not only create higher quality code but also cut expenses. When using a serverless model, you're only charged for the CPU cycles and memory you actually use.

Enhanced scalability

Many business owners want their apps to become influential and scalable like Google or Facebook. Serverless computing makes scaling automatic and seamless. Your app will automatically scale as your load or user base increases without affecting performance. Serverless apps can handle a huge number of requests, whereas a traditional app will be overwhelmed by a sudden increase in requests.

Cons of a serverless architecture

Vendor lock-in

Vendor lock-in describes a situation when you give a vendor full control of your operations. As a result, changes to business logic are limited and migration from one vendor to another might be challenging.

Not for long-term tasks

A serverless model isn't suitable for long-term operations. Serverless apps are good for short real-time processes, but if a task takes more than five minutes, a serverless app will need additional FaaS functionality.

The bottom line

Serverless software architecture is beneficial for accomplishing one-time tasks and auxiliary processes. It works great for client-heavy apps and apps that are growing fast and need to scale limitlessly.

And finally, let's look at the following image to know when to choose each of these four architecture types:

Choosing the right architecture for your MVP is a daunting task, but RubyGarage has many years of experience to help you with your choice. We would be glad to answer all your

questions on this topic.

TAGS:

Web MVP

AUTHOR:

Anastasia D.
Copywriter

THANK YOU! WE WILL TAKE YOUR APPRAISAL INTO ACCOUNT SOON.

27 rating, average 4.7 out of 5

SHARE ARTICLE WITH

COMMENTS (6)

to leave a comment Sign in

Mohamed Fouad over 2 years ago

Very good comparison

Reply

Maryna Z. over 2 years ago Mohamed Fouad

Thanks for appreciating our work!

Reply

Dinesh Kumar over 2 years ago

Thanks a lot for your content & I really appreciate that.

Reply

Anastasia Z. over 2 years ago Dinesh Kumar

Thanks for appreciating our work!

Reply

Dhurkeeswaran K about 2 years ago

Great work... Very neat explanation. Thanks a lot :)

Reply

Muhammad Husnain 10 months ago

this was one of the best blog that i ever read. Amazing work.

Reply

RECOMMENDED ARTICLES

Three Database Architectures for a Multi-Tenant Rails-Based SaaS App

Technology Stack for Web Application Development

What Benefits Can a Microservice Architecture Bring to Your Project?

ENTREPRENEURSHIP 115103 VIEWS 9 MIN

INSIGHTS 435234 VIEWS 14 MIN

ENTREPRENEURSHIP 18335 VIEWS 6 MIN

RubyGarage

+1 (929) 999-19-70

+44 (208) 068-49-88

Featured in

Forbes

Inc.

Apple News

HUFFPOSTI

HACKERNOON

Case Studies

Blog

Contacts

Top Rated in

Clutch.co

Services

Product Ideation

Services

Design

Web Development

Mobile Development

Software Testing

Source Code Audit

For Enterprise

For Startups

Industries

Retail

Fintech

Education

Healthcare

Company

About

Workflow

Engineering

Careers

Locations

Boston

Chicago

Houston

Manchester

New York

San Francisco