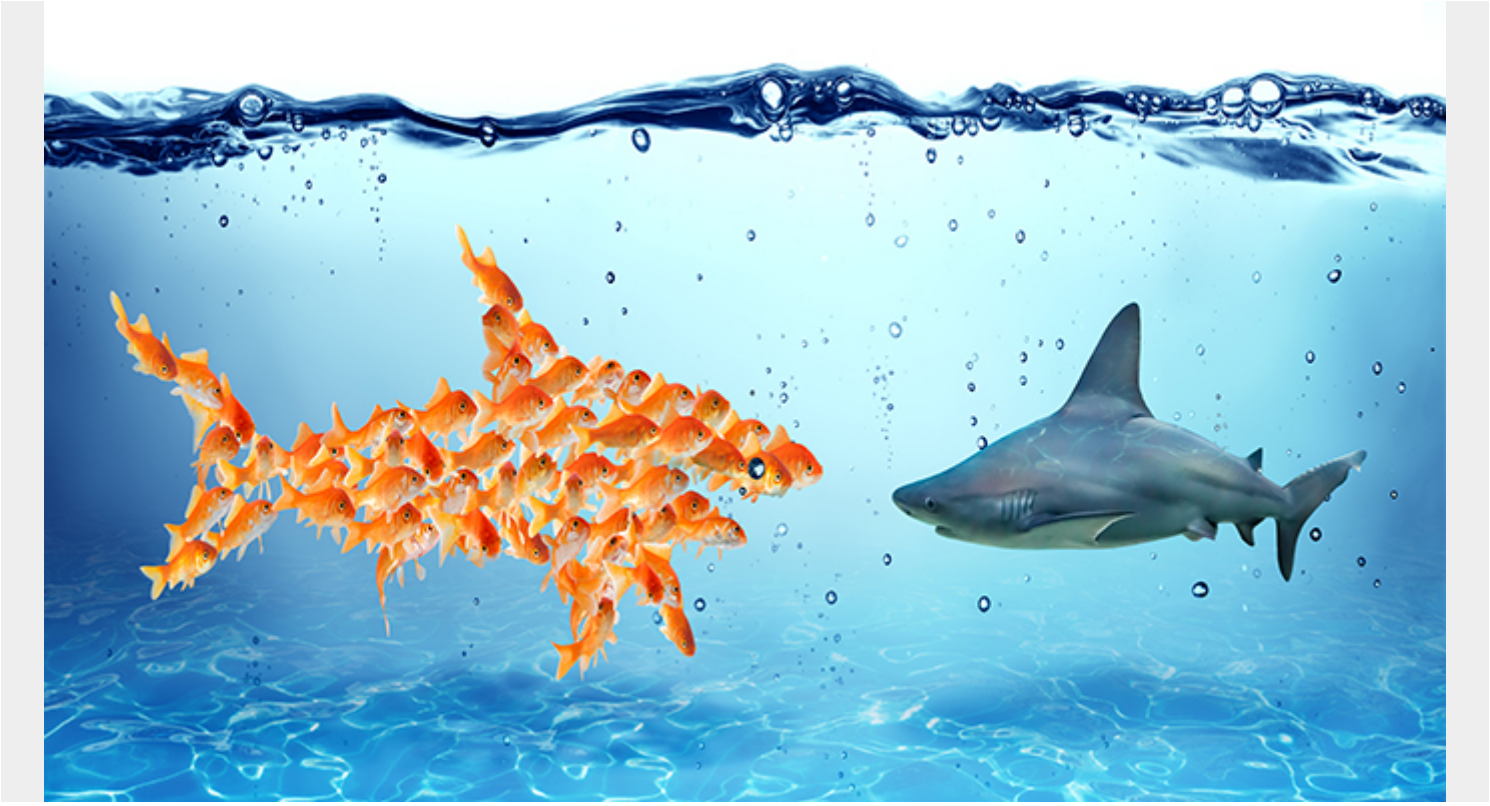
WHAT IS MICROSERVICE ARCHITECTURE? MICROSERVICES EXPLAINED



Microservice architecture, aka microservices, are a specific method of designing software systems to structure a single application as a collection of loosely coupled services. Applications tend to begin as a monolithic architecture (more on that below), and over time grow into a set of interconnected microservices.

The main idea behind microservices is that some types of applications are easier to build and maintain when they are broken down into many small pieces that work together. Though the architecture has increased complexity, microservices still offer many advantages over the monolithic structure.

The concept of *micro* stems from the existing monolithic infrastructure most companies came up using, especially if the company has been around for a decade or longer. Instead of a monolithic architecture, each component of a microservice architecture has:

- Its own CPU
- Its own runtime environment
- Often, a dedicated team working on it, ensuring each service is distinct from one another

This architecture means each service can:

- Run its own unique process
- Communicate autonomously without having to rely on the other microservices or the

application as a whole

This ability to be separated and recombined protects the entire system against decay and better facilitates agile processes, making it appealing to organizations—especially those still utilizing monolithic infrastructures.

This article will cover:

- How microservices work
- Examples
- Monolithic vs microservice architectures
- Pros & cons of microservices
- Helpful resources

How microservices works

Microservices are a set of services that act together to make a whole application operate. This architecture utilizes APIs to pass information, such as user queries or a data stream, from one service to another.

How the underlying software works, or which hardware the service is built upon, depends solely on the team who built the service. This makes both communicating between teams and upgrading services very dynamic—even reactive—allowing a software company or team to be more resilient in its development.

Kubernetes has helped advance the cause of microservices, though it not a necessary building block. The rise of cloud computing and networked computers has done two things:

- Removed the responsibility from the user needing to have a powerful computer to run all the necessary operations.
- Places the responsibility on the company to use individual servers to run its service each time a user runs the application.

In the case of microservices, the user's machine may be responsible for basic processing, but it is mostly responsible for sending and receiving network calls to other computers.

Whenever you use an application, it's reasonable to assume that there are five other computers, give or take, that just turned on in order to power your experience. In the case of something like Facebook or Uber, it may be more reasonable to expect another 10,000 computers are actively processing information to enhance the user experience.

Microservices are often considered a logical evolution of Service Oriented Architecture (SOA), but there are clear differences between the two.

Examples of microservices architecture

As Martin Fowler points out, many large companies now utilize microservices within their architecture. Netflix is one of the earlier, most well-known adopters. Some other well-known examples are:

- eBay
- Amazon

- Twitter
- PayPal
- SoundCloud
- Gilt
- The Guardian

Present in each of these companies are a network of microservices.

For example, SoundCloud might have a new user microservice designed to onboard a user onto its application. The microservice will activate the user's account on the backend, and it might be responsible for sending a welcome email to the user, and a walkthrough when the user first logs into the application.

Another microservice for Soundcloud might be to handle uploading and storing a user's song to the platform. Another might be its search functionality and recommended artists.

As a company is divided into departments with people having different responsibilities, like a sales agent, a financier, and a bank teller are all points of contact with the same bank, a company's microservices divide the responsibility of the whole company into individual actions.

The difference between this microservice design and a monolith is that Soundcloud does not have one single application to handle each of these parts, released in the spring of each year and distributed on a CD-ROM, for example. Instead, each part (microservice):

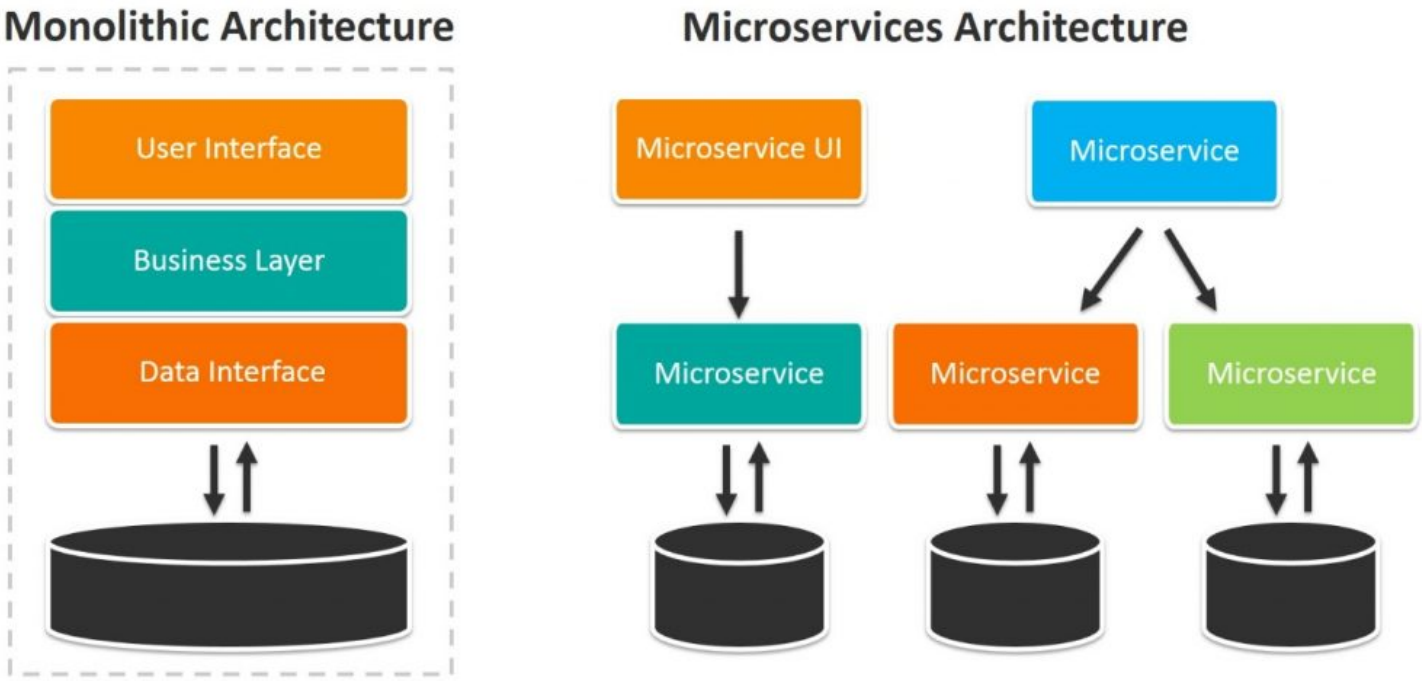
- Works autonomously to contribute to the whole
- Can be upgraded in the modern way of continuous development and integration (CI/CD)

(Explore microservices best practices that will help you get up and running successfully.)

Monolithic architecture vs microservice architecture

The monolithic architecture pattern has been the architectural style used in the past, pre-Kubernetes and cloud services days.

In a monolithic architecture, the software is a single application distributed on a CD-ROM, released once a year with the newest updates. Examples are Photoshop CS6 or Microsoft 2008.



That style was the standard way of building software. But as tech has evolved, so too the architectural style must advance. In an age of Kubernetes, and CI/CD workflows, the monolithic architecture encounters many limitations—companies need to push to microservices.

Characteristics of a monolithic architecture:

- Changes are slow
- Changes are costly
- Hard to adapt to a specific, or changing, product line

Monolithic structures make changes to the application extremely slow. Modifying just a small section of code can require a completely rebuilt and deployed version of software.

(Learn about the complexities of app modernization and code refactoring.)

If developers wish to scale certain functions of an application, they must scale the entire application, further complicating changes and updates. Microservices help to solve these challenges.

Advantages to Microservices

Applications built as a set of independent, modular components are easier to test, maintain, and understand. They enable organizations to:

- Increase agility
- Improve workflows
- Decrease the amount of time it takes to improve production

While each independent component increases complexity, the component can also have added monitoring capabilities to combat it.

Here are the most common pros of microservices, and why so many enterprises already use them.

Developer independence

Each microservice will often be assigned a single dev team to maintain it. Thus, there is greater developer freedom and independence.

Small teams that are working in parallel can iterate faster than larger teams. When a single service takes off in popularity, the smaller team can also scale the services on their own without having to wait for a larger and more complex team.

Isolation & resilience

If one of the components should fail, due to issues like outdated technology or inability to further develop the code, developers are able to spin up another component while the rest of the application continues to function independently.

This capability gives developers the freedom to develop and deploy services as needed, without having to wait on decisions concerning the entire application.

Scalability

Because microservices are made of much smaller components, they can take up fewer resources and therefore more easily scale to meet increasing demand of that specific component.

As a result of their isolation, microservices can properly function even during large changes in size and volume, making it ideal for enterprises dealing with a wide range of platforms and devices.

Autonomously developed

As opposed to monoliths, individual components are much easier to fit into continuous delivery pipelines and complex deployment scenarios. Only the pinpointed service needs to be modified and redeployed when a change is needed. If a service should fail, the others will continue to function independently.

Its autonomous nature benefits teams because it:

- Enables scaling and development
- Doesn't require much coordination with other teams

Microservices are a particular advantage when companies become more distributed and workers more remote.

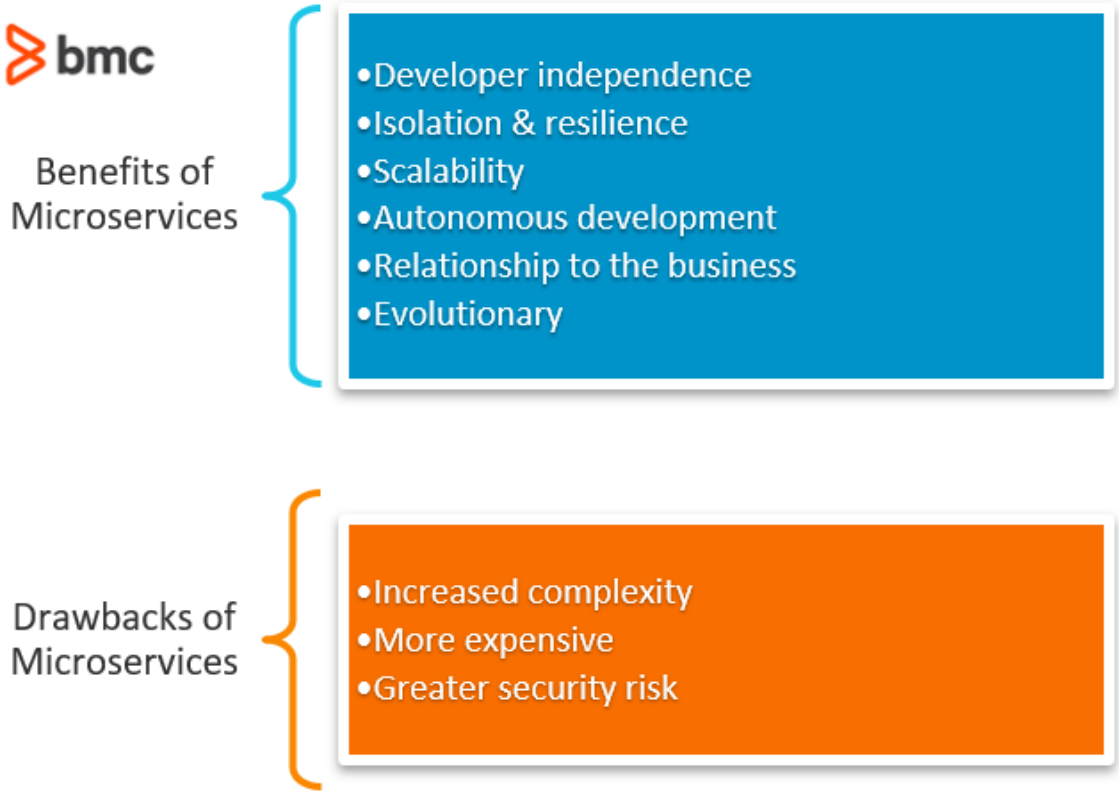
Relationship to the business

Microservice architectures are split along business domain boundaries, organized around capabilities such as logistics, billing, etc. This increases independence and understanding across the organization: different teams are able to utilize a specific product and then own and maintain it for its lifetime.

Evolutionary

Any microservice architecture is highly evolutionary.

Microservices are an excellent option for situations where developers can't fully predict what devices will be accessed by the application in the future. They also allow quick and controlled changes to the software without slowing the application as a whole—so you can be more iterative in developing features and new products.



Disadvantages of microservices

Of course, the microservice architecture comes with a learning curve. First-time users might struggle to determine:

- Each microservice's size
- Optimal boundaries and connection points between microservices
- The right framework to integrate services

More broadly, microservices have these drawbacks:

- Increased complexity
- More expensive
- Greater security risks

Increased complexity

First and foremost, microservices are a much more complicated system. It has a learning curve that can be steep to climb, but once learned, as with most things, it can be used with ease. New solutions are not always better, and it is the fault of many consultants and young engineers who try to solve different problems with the same solutions.

The microservice architecture is not always the best solution for an application. For some it might be too complex than what is required.

More expensive

Microservices can also be more costly. They usually run in their own environments with their own CPUs. They work through API calls which have a price tag. And, finally, the more complicated the environment requires a team of engineers capable of building it, so labor costs are going to be more.

Greater security risks

Finally, because microservices will exist through different environments on different running machines with different API calls, they offer more points of contact for an attacker to get in and damage the system.

Microservices support DevOps

In the end, microservices are part of the comprehensive shift to DevOps that many organizations are working towards. DevOps environments promote strong collaborations between development and operations teams to make IT become more agile—so the business is agile, too.

Microservices are merely a technology to use. Instead, they're part of a larger concept that organizations need to adopt culture, knowledge, and structures for development teams in support of business goals.

Related reading

- BMC DevOps Blog
- Microservices vs Miniservices: Choosing the Right Framework
- Microservices vs Serverless: What's The Difference?
- Kubernetes Guide, a series of articles and tutorials
- Kubernetes vs Docker Swarm: Comparing Container Orchestration Tools
- DevOps Engineer Roles & Responsibilities