MICROSERVICES VS NANOSERVICES: WEIGHING FRAMEWORK OPTIONS

Over the years, software development moved away from traditional monolithic architectures, addressing its complexities with tightly coupled, interconnected code. This resulted in the adoption of microservices, efficient cloud-native approaches that enable distributed computing through multiple, smaller services.

Since microservices have become mainstream, in the last several years, nanoservices evolved as another pattern that was designed mostly to overcome complexities found with microservices. Essentially, nanoservices divide the services further to bring additional efficiency.

This article outlines how developing smaller components to build an application is a preferred approach, so I will:

- Explain both micro and nanoservices
- Explore benefits and challenges
- Share examples and best practices to help you determine the right framework for your needs

Improving development practices

Due to their elemental design, monolithic applications have a few serious challenges:

- They're cumbersome.

- They use a lot of disk space.
- They're hard to scale.

When monolith applications are mature, introducing the latest technologies or adopting enhanced frameworks comes with a lot of intricacies. These obstacles are not limited to the code-level integration. Testing and deployment are additionally challenging.

Besides, an entire project to rewrite the whole application is often time-consuming and highly risky.

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

Microservices over monoliths

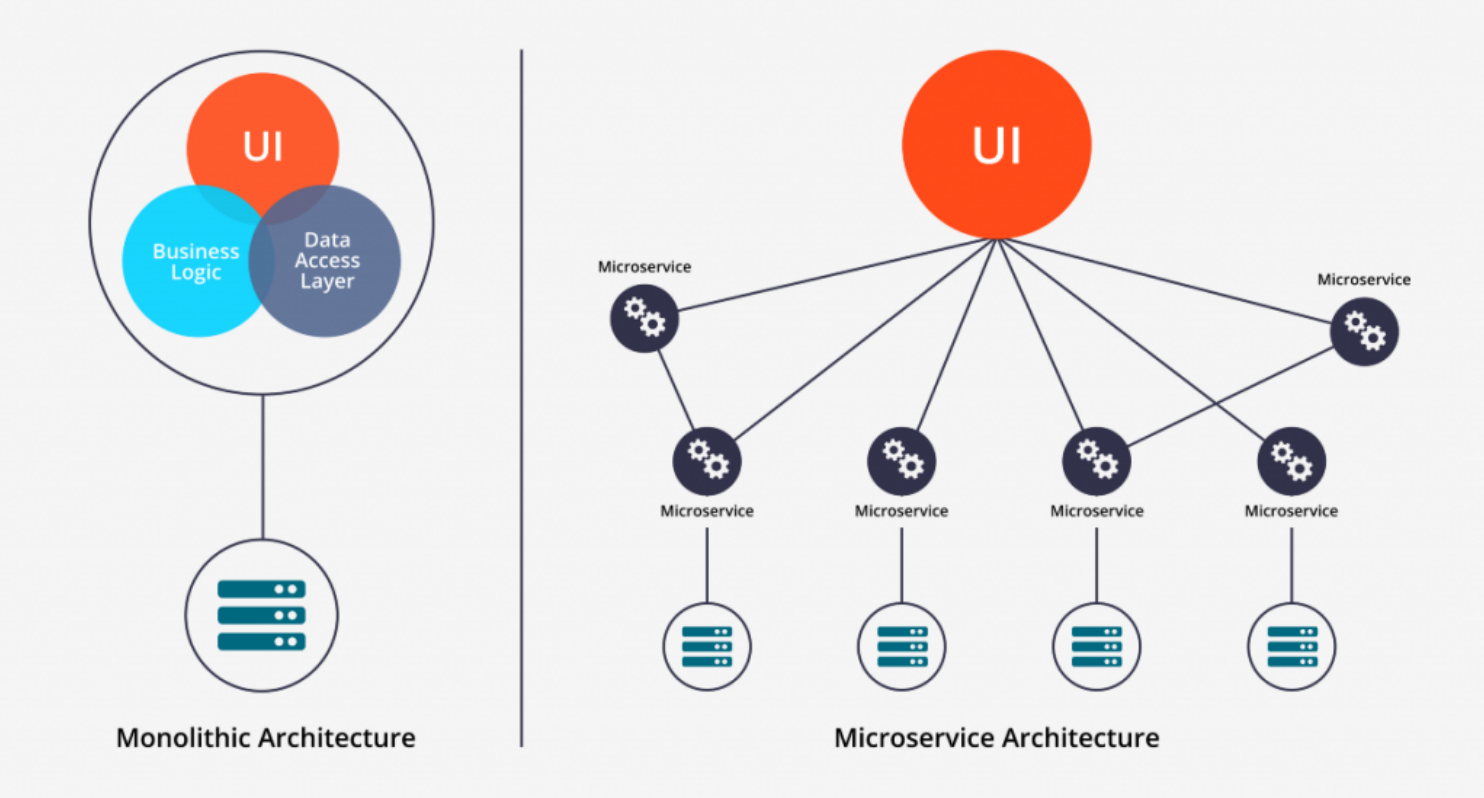
Microservices were constructed to:

- Overcome the problems with a monolithic approach.
- Provide an effective solution for developing distributed, cloud-based applications.

A microservice fragments an application into an assembly of multiple loosely coupled services. Typically, microservices are:

- Developed and tested in isolation
- Organized by the processes they handle
- Deployed independently.

Each of these services has its own separate processes and is designed to function independently without sharing resources in order to avoid conflicts.



(Source)

A typical illustration shows how a central, multi-tier framework of a monolith differs from a microservices architecture with distributed services.

(*Compare microservice architecture to monolithic architecture.*)

Microservices are usually contained inside cloud-based containers that communicate with other services through web APIs. Such APIs act as messaging queues that respond to incoming events such as, user interface, invoking other services, accepting data, etc.

Due to their distributed model, microservices are developed and maintained without impacting other services. This essentially also aids a DevOps model by allowing:

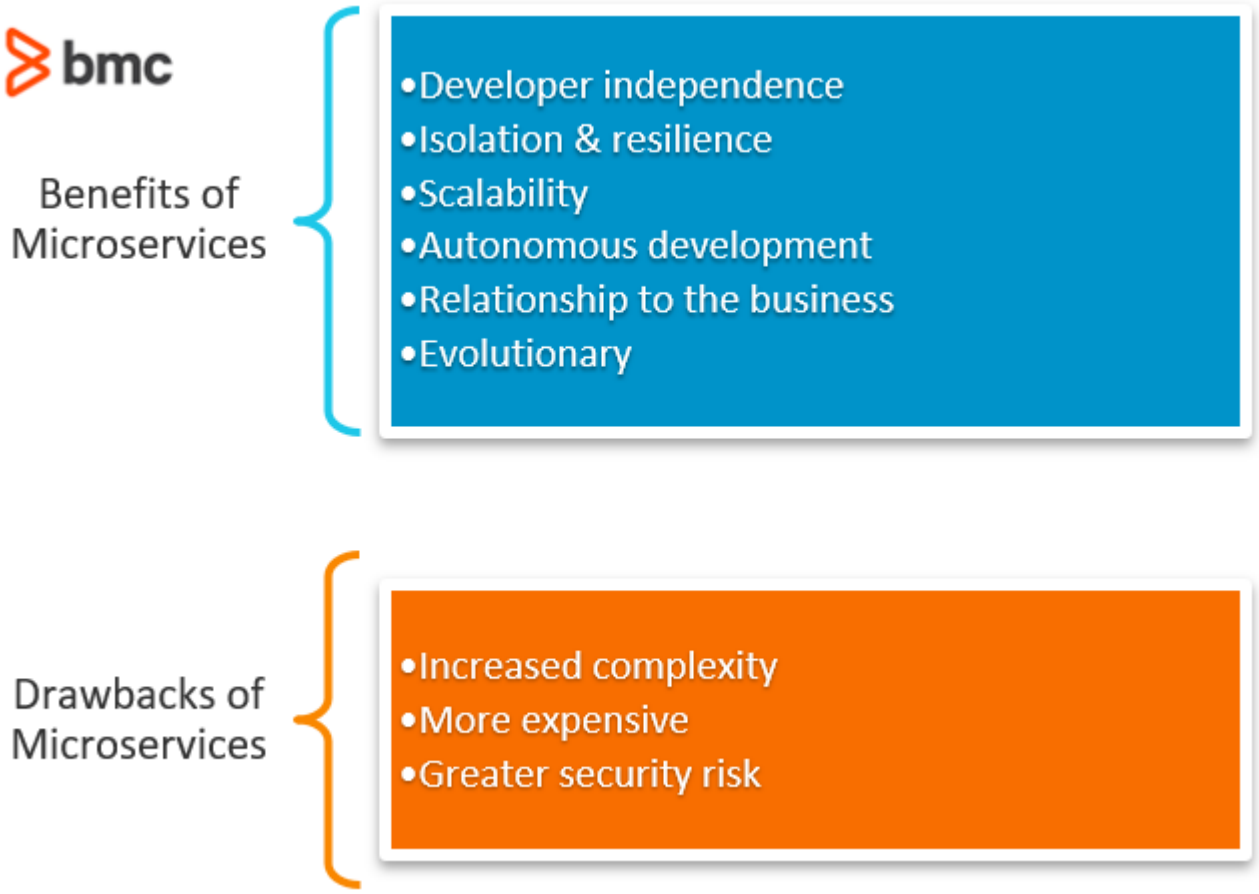
- Efficient automation
- High availability

(See how microservices and APIs differ but work together.)

Benefits of microservices

The component-based design of a microservice framework offers multiple benefits, including:

- Microservices are easier to maintain, scale, and share
- They work well with containers.
- Development and testing is quicker, shortening the deployment cycle and making it an ideal fit for DevOps practices.
- The isolated nature of services offers safer deployment and quicker resolution of errors.
- Microservices provide greater flexibility by allowing reusability to build similar services.
- There are no restrictions on the specific technologies to use, allowing greater development productivity (and perhaps innovation).
- Because they're interoperable in nature, microservices can be used with both open-source or closed-source platforms.



What are nanoservices?

The basic difference between the two frameworks is that nanoservices are considered smaller siblings of microservices. Nanoservices are designed to perform a single function, whose output is exposed through a specific API endpoint (command).

Nanoservices are fully discoverable among each other. Each one can link with other services to perform additional actions and extend functionality.

By design, nanoservices are:

- Self-contained
- Reusable
- Less complex than microservices
- Supported by distinct function files for each

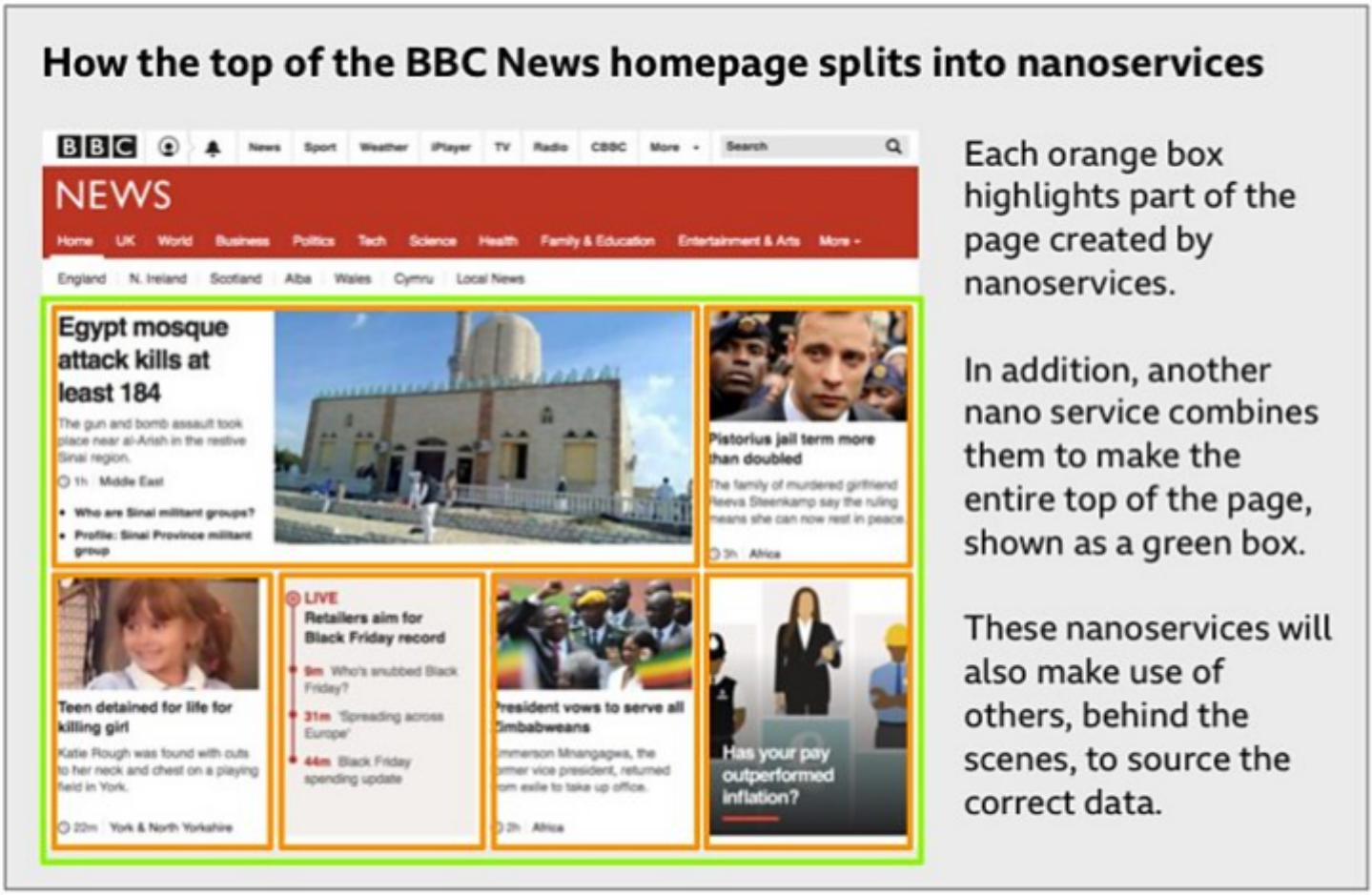
Nanoservices are considered more efficient than microservices, particularly due to:

- Ease of initial setup
- Reduced latency among services, where each nanoservices' function code can be executed on-demand without the use of dedicated containers or servers to host every individual service.

Nanoservice real world use case

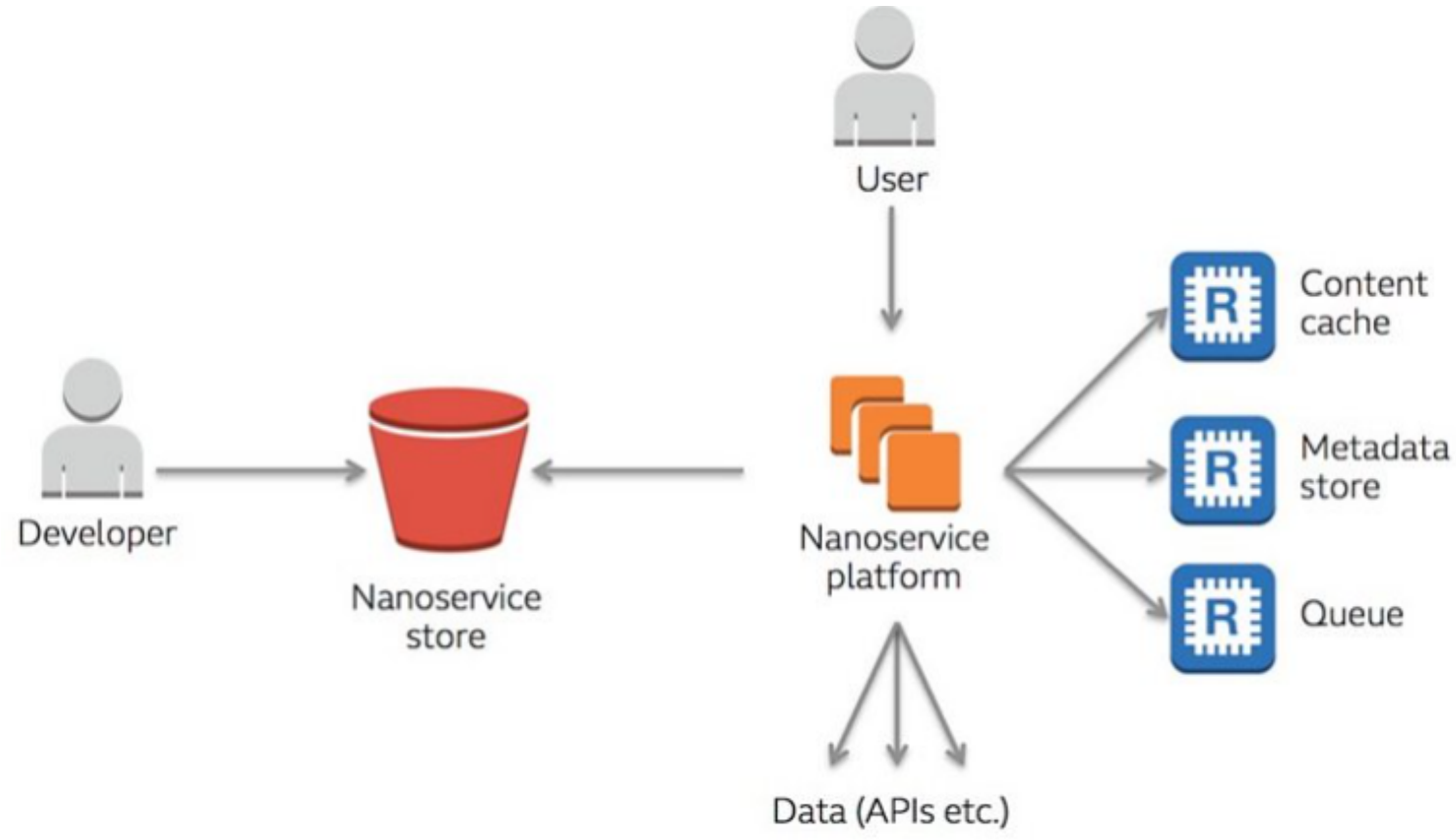
The illustration below highlights an interesting use-case of how the BBC website uses 1,000s of nanoservices to render dynamic web pages. Each service functions to feed into a specific component of the webpage, such as:

- Generating a headline
- Sourcing weather data
- Updating a match score
- Etc.



(Source)

A high-level nanoservice architecture of BBC's site outlines how multiple, smaller services function in conjunction to deliver content of a webpage:



Benefits of nanoservices

Nanoservices are particularly beneficial due to their ease of encapsulation of specific functions. This brings an application framework with these advantages:

- More focused and lightweight compared to microservices
- Multiple teams can work on a specific service or multiple services
- Each service can have its own security protocol
- Can be developed, tested, and deployed independently
- Facilitates faster and more frequent code releases
- Added flexibility to link different services together for extending functionality

Current challenges

While microservices and nanoservices are significant enhancements over a monolith, they don't come without challenges.

Certain use cases also suggest that developing an application by combining smaller services increases complexity in a number of ways, including:

- More effort spent on creating and maintaining duplicate services
- The requirement of a highly efficient infrastructure is complex to build and costly to maintain
- Higher resource consumption, due to the sheer number of services you're supporting
- Tighter service-level dependencies may result in complex error-tracking

At the first glance, increased isolation within nanoservices may seem tempting. But, widespread use of nanoservices could lead to a whole host of problems and increased complexities.

More so, for decently complex applications, adopting a nanoservice framework may actually overutilize resources and scale up manual efforts—something that DevOps practices want to minimize.

(Prepare for other common microservice challenges.)

Choosing micro or nanoservices

Both microservices and nanoservices have evolved to bring efficiency over traditional architectures, by distributing complex application codes into multiple smaller services within a cloud-native setup.

With that in mind, the important considerations of choosing the best framework eventually come down to two key components:

- Developer experience
- The application's functionality requirements

When to use microservices

Compared to nanoservices, microservices are an established and well-known design approach that can operate with a wide range of technologies.

There are several successful use-cases of major industry players—Netflix and AWS—who already use microservice architecture to offer several state-of-the-art services. That means microservices have a plethora of resources and groups to support learning or maturity microservices strategy.

Microservices are usually a good fit for:

- An agile development process
- Situations where development teams are functionally or geographically distributed
- Creating a minimum viable product (MVP) that you will extend functionally over time, not all at once

With that in mind, it is equally important to note that microservices are not suited for every application type. Certain use cases highlight that, without thorough due-diligence and efficient planning, random expansion of services may end up making the framework as big and cumbersome as a monolithic architecture.

(Follow microservice best practices from planning all the way through maintenance.)

Nanoservices

Nanoservices are much newer. Proven use cases of their full potential are yet to be discovered. While they are smaller, flexible, more isolated, and functionally focused, there are still unknowns on:

- Compatibility with emerging tech
- Exponential scaling

As a result, there are conflicting schools of thought around how and when to use microservices.

First, the mere nature of nanoservices poses the question as to whether they are secured from attack vectors when used extensively in an application.

Some also consider nanoservices to be an anti-pattern due to their fragmented functionality. This is known to cause maintenance overhead to exceed the value of the framework's benefits.

As nanoservices are still in their infancy, conservative and large organizations usually avoid them (for now), mostly due to the framework's uncertain support for scalability. For those who adopt it, one best practice of using the framework is:

Avoid nanoservices if its use complicates the system that it's meant to simplify.

Designing intentionally

Whether to choose nanoservices over microservices, or vice-versa, will always differ with use-cases. The end goal, however, remains the same: to adopt the right architecture that aids automation and helps develop applications that are resilient, scalable, and secured.

Related reading

- BMC DevOps Blog
- Microservices vs Miniservices: Choosing the Right Framework
- Microservices vs Serverless: What's The Difference?
- Kubernetes vs Docker Swarm: Comparing Container Orchestration Tools, part of our Kubernetes Guide
- Agile vs Waterfall SDLCs: What's The Difference?
- Top DevOps Trends Today