MICROSERVICES VS SERVERLESS: WHAT'S THE DIFFERENCE?

Both serverless and microservices technologies are designed with the goal of hosting highly scalable solutions. But, they aren't the same thing.

In this article, we'll take a high-level view of each technology so you can determine what's best for your application development and your overall business.

A brief history of app development

Like anything in technology, microservices and serverless are the response to a particular problem. It all begins with traditional application building.

The traditional approach to building a web-based application has been to implement a monolithic architecture. In its most basic form, a monolithic architecture consists of:

- A central database
- A web server
- A user interface (the browser)

This required two critical components:

- Hosting the physical database and web server on-premises
- Employing an in-house technical team to maintain the architecture

As businesses scaled up, monoliths turned out to be unscalable. They were also less adaptable, more resource-intensive, and overall more expensive to manage due to high technical overhead.

To deal with this, companies needed an architecture that was agile, scalable, and cost-

effective—while still delivering on performance.

These requirements gave birth to serverless and microservices technologies, which are designed with the goal of hosting highly scalable solutions.

While both support a modernized approach to IT and developer operations, each comes with its own set of advantages and challenges.

What are microservices?

Building a microservice architecture (MSA) consists of several *autonomous* components that interconnect with each other using APIs. Each of these components—known as microservices—executes a single function or process. Each microservice is deployed within a container that operates as a stand-alone application.

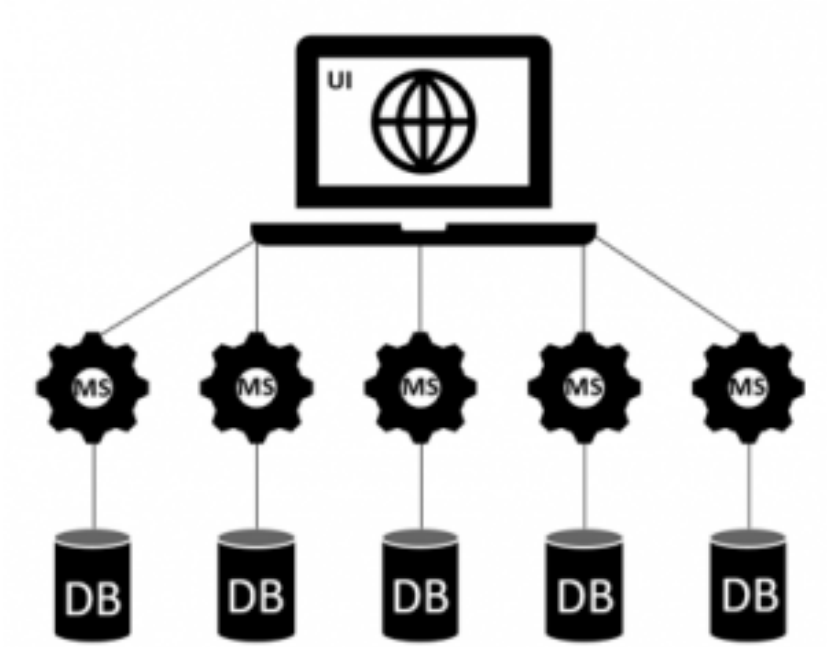
Essentially, each microservice contains basic elements that support its independent run-time, such as:

- Its own database
- Libraries
- Templates

Typically, a DevOps team breaks down all features and functions of a software application into separate services in such a way that the app's entire functionality is retained while converting it to a decentralized model. The team defines the dependencies between these components, then assigns a component to different teams of developers for development and maintenance.

While doing so, each of these microservices is independently developed and tested before being deployed in isolated containers.

This approach helps form a framework where even if one microservice breaks or undergoes maintenance, it is easier to fix and redeploy without impacting other services or the overall framework.



Independent microservices working with their own database to execute functionality requested by the user

interface.

When to use microservices

Microservices are best suited for applications and systems that are constantly evolving, complex, and require high scalability.

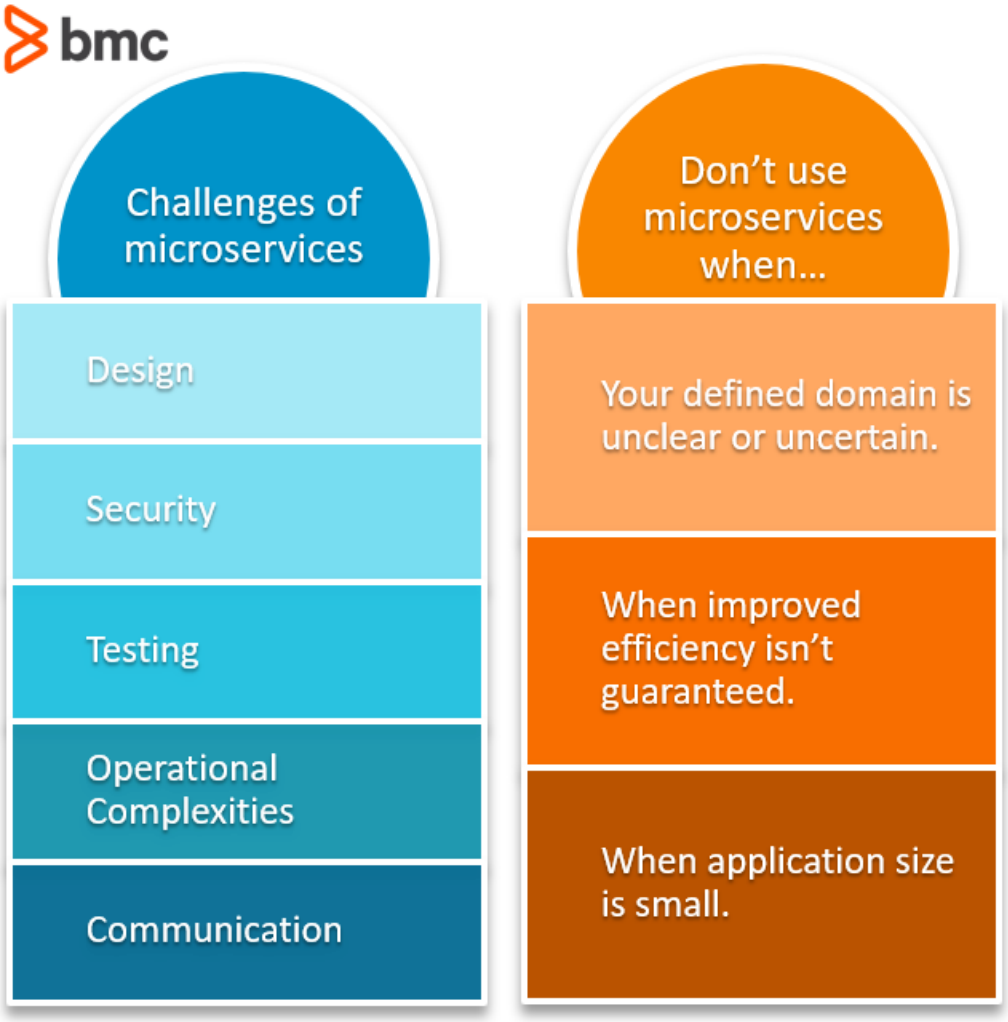
Particularly for applications that involve the processing of large amounts of data, microservices are ideal because you can break down complex functions into multiple services for easier development and maintenance.

Benefits of a microservice framework

- Ability to independently develop, test, and deploy services without affecting other services or the parent application
- Isolating faults within larger applications is easier
- Increased agility through quicker development and issue resolution without relying on the release of the entire application
- Easily scale performance by adding microservices during traffic spikes
- Flexibility to change business logic and use a wide range of technologies
- Can be adapted and reused in multiple processes or different contexts

Challenges of a microservice framework

- Complexity increases when split into autonomous components
- Increased overhead of managing multiple databases, ensuring data consistency, and constantly monitoring each microservice
- Microservice APIs are four times more vulnerable to security attacks
- The need for expertise and computing resources can be expensive
- Can be too slow and complicated for smaller companies that need to quickly implement and iterate
- A distributed environment needs tighter interface controls and high test coverage



(Learn more about common challenges with microservices.)

Now, let's turn to serverless.

What is serverless?

The term serverless is often misleading: it implies there is no server involved.

What serverless actually means is that an organization does not need to invest in or maintain physical hardware. Instead, you rely on a trusted third-party to manage the maintenance of the physical infrastructure, including the server, network, storage, etc.

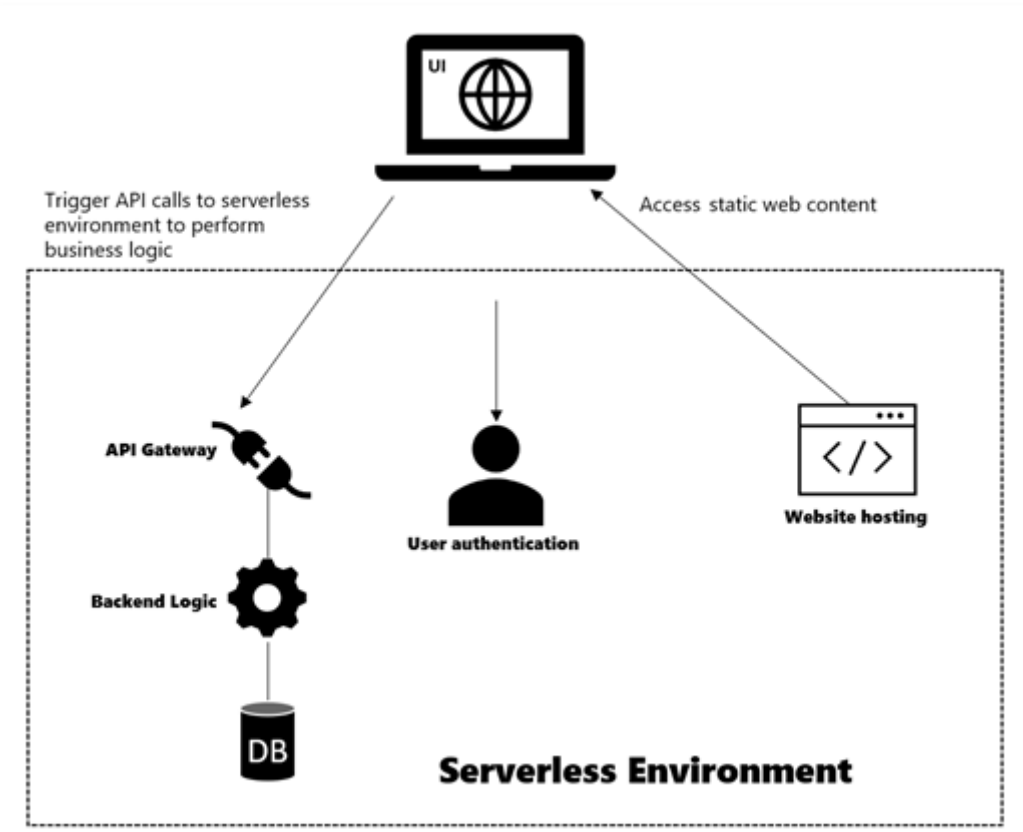
This approach lets your organization develop applications without needing to manage the underlying infrastructure. Popular third-party serverless platforms include:

- AWS Lambda
- Microsoft Azure Functions
- Google Cloud Platform Functions

Serverless includes two different perspectives:

- Function as a Service (FaaS). An evolved model that allows developers to run code module (functions) of an application on the fly, without getting concerned about the backend infrastructure or system requirements.
- Backend as a Service (BaaS). A model where the entire backend (database, storage, etc.) of a

system is handled independently and offered as a service. This usually involves outsourcing backend services to a third-party for maintenance and management, leaving your organization to focus on developing your core functions.



Different components of a serverless framework interacting with the UI function.

Benefits of a serverless environment

- More focus on developing quality applications with quicker deployment
- Less time and expense spent on building, maintaining, and upgrading the underlying infrastructure
- Best suited for short and real-time processes that are client-heavy and are expected to grow
- Reduced costs in hiring database and server experts
- Multiple subscription-based pricing models for efficient budget projection
- Quick scalability—without affecting performance
- Responsibility and management of computing resources lies with the vendor, not you

Challenges of a serverless environment

- Commitment to long-term contracts with the managing third-party
- Business logic or technology changes can make transitioning to another vendor challenging
- Multi-tenant serverless platforms may introduce performance issues or bugs within a pooled platform if/when a neighboring tenant deploys faulty code
- Applications or functions that are inactive for a prolonged period may require a cold start to run, which takes added time and effort to initialize resources

Microservices vs serverless: which one is right for you?

Of course, there are advantages and disadvantages of both microservices and serverless architectures. Determining which architecture to go with comes down to analyzing your company's business goals and product scope.

Ultimately, if cost and a quick deploy-to-market are priorities, then serverless is a good bet. However, if your organization intends to build a large, complex application where the product is expected to evolve and change, then microservices is a more practical option.

Alternatively, with the right team and effort, it is also possible to combine both these solutions within a single cloud-native instance.

Related reading

- BMC DevOps Blog
- 15 Best Practices for Building a Microservices Architecture
- Microservices vs SOA: How Are They Different?
- The State of Serverless Today
- Deploying vs Releasing Software: What's The Difference?