SERVERLESS ARCHITECTURE: THE BEGINNER'S GUIDE



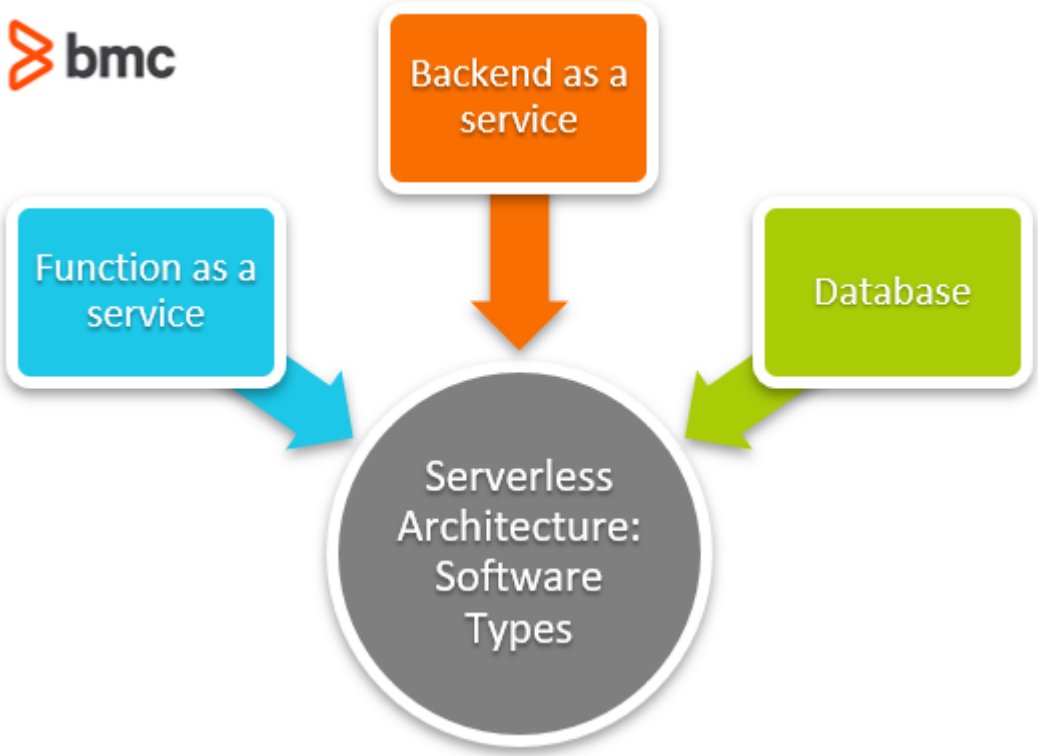
Serverless architecture refers to the software design pattern where infrastructure management tasks and computing services are handled by third-party cloud vendors through functions. These functions are invoked and scaled individually during the software development process. The vendor handles these in their entirety:

- Physical hardware-related concerns
- Several infrastructure and operations tasks

Serverless architecture is different from serverless computing. Serverless computing can be used to write functions that include resource configuration changes necessary to perform certain individual infrastructure management tasks.

Types of serverless architecture software

There are three primary services offered via software developed with serverless architecture:



Function as a service

In the realm of pre-packaged services, function as a service, sometimes known as FaaS or framework as, falls in between software as a service and platform as a service.

Think of FaaS as a ready-to-implement framework that can be easily tailored to the needs of an enterprise company. To be clear:

- SaaS is ready to use out of the box while FaaS is not.
- However, FaaS does not require the resources to implement that you would need if you were using PaaS.

FaaS can be delivered in customizable templates, for instance, by industry verticals. FaaS uses containers to prime for rapid deployment of applications on all platforms. For instance, developers can stack containers for scalability or write one container for iOS development and another for desktop applications.

Consumers purchase FaaS from third-party vendors who handle server management. They are then charged for actual runtimes instead of pre-allocated units. Companies who use FaaS benefit from improved efficiency as fewer resources are spent on rapid development of applications.

(Compare FaaS & serverless.)

Backend as a service

Similar to FaaS, backend as a service (BaaS) is another serverless technology. Some will contend that BaaS takes it a step further as a NoOps offering. NoOps essentially refers to infrastructure that has been automated to the point that in-house developers have no hand in its operation.

Here's an easy way to look at BaaS: Imagine your enterprise organization is developing a mobile app to connect employees to important information on the go. You might develop the basic application

framework in-house and then outsource the functionality. This includes backend processes like:

- Accessing cloud storage
- Syncing
- Social collaboration

A company's ability to offer backend services that link a mobile application to the cloud is referred to as BaaS.

Database

Database serverless frameworks access and automate your database functions. These are functions that write and read from a database and can also provide a response.

Serverless database frameworks offer companies room to grow globally, as multiple applications can be developed by region. Still, they all run from one location, powered by FaaS technology.

Serverless design principles

The common design principles of serverless architecture applications include:

- **Simplicity & speed.** Write functions that are concise and designed to perform unique transactional operations—complete computing tasks performed on one or more entities. These transactions have time and capacity limitations and are therefore designed for fast execution.
- **Optimized for concurrency.** Design function requests based on concurrency limitations of the serverless architecture. The total request count may peak less frequently as compared to concurrent request capacity limits and optimizing for total request may not be the optimal design approach for serverless applications.
- **Temporary storage.** When functions are used, the underlying resources are provisioned or accessed for a limited duration. Function execution may involve changing states of the environment, including storage capacity, and therefore using a persistence may be preferable to address durable storage requirements.
- **Hardware agnostic.** Since the resources are provisioned for the duration of the function runtime, it's important to eliminate hardware related dependencies on serverless applications.
- **Iterative state machine orchestration & events.** Application architecture should be orchestrated based on the output of iterative transactional operations. Hardcoding workflows introduce potential dependencies that cause the serverless application to be tightly coupled.
- **Redundancy.** Design for failure. Especially for downstream calls, one failure can channel the failure to subsequent requests and impede the operational workflow of the application.

(Understand the impact of redundancy on availability.)

Advantages of serverless architecture

With these characteristics, serverless architecture offers the following value propositions:

- **Deploy & run.** The infrastructure resources are managed by the cloud vendor. Internal IT can therefore focus on the business use case of software applications instead of managing the underlying hardware. Functions allow users to deploy application builds and configuration files necessary to provision the required hardware resources.

- **Fault tolerant.** Since serverless application coding is logically decoupled from the underlying infrastructure, hardware failures have minimal impact on the software development process. Users are not required to manage applications on their own.
- **Low operational overhead.** The infrastructure and operations management tasks are managed by cloud vendors, allowing organizations to focus their efforts on building software features. Applications are released faster, resulting in faster end-user feedback and therefore, continued improvements over the next software release cycles.
- **Optimized usage-based billing.** The pay-as-you-go billing model serves particularly well for small and midsize (SMB) organizations that lack the capital to establish and manage on-site data centers. The ongoing OpEx model is further optimized as functions are configured against potential over-provisioning and inadequate resource utilization.
- **Built-in integrations.** Most cloud vendors offer integrations with a variety of services that allow users to focus on building high quality applications instead of configuring them.

When to avoid serverless architecture

When it comes to building cloud-based applications, serverless architecture is not always the best choice. In fact, you might want to skip serverless architecture altogether when it comes to these situations:

- **Fixed architecture applications.** Software that require the underlying architecture to be explicitly maintained and unmodified. Security sensitive and offline software use cases in the defense and financial industries often face these requirements to facilitate high application dependability.
- **Tightly controlled environments.** Use cases that require tight control on hardware and lower layers of the IT network, such as installing specific components, patches, and hardcoding configuration changes.
- **Low latency applications.** While serverless architecture functions are designed for speed at the system-level, not all incoming requests and transaction operations are handled efficiently.
- **Pre-migrated applications.** For legacy applications that are yet to move to the cloud environment, using a serverless architecture offers limited business value, since internal IT has to manage the hardware regardless of the choice of application architecture.
- **Application performance limitations.** Serverless functions face limits in terms of capacity, speed, and runtime. The performance of serverless functions should match or exceed the capacity limits of serverless services such as AWS Gateway timeout.

Because of these challenges, serverless architecture is usually less than ideal for high-performing, complex application builds.

Related reading

- BMC DevOps Blog
- Serverless Best Practices
- Serverless vs Platform as a Service: Is Serverless the New PaaS?
- Microservices vs Serverless: What's The Difference?
- Best Practices to Avoid Cloud Vendor Lock-In