SERVERLESS VS PLATFORM AS A SERVICE: IS SERVERLESS THE NEW PAAS?

The increased demand for faster and flexible software development has put organizations on a constant lookout for new technologies and platforms to cope with the demand.

Solutions like Platform as a Service (PaaS) have gained popularity by providing complete software development and deployment platforms in the cloud—and reducing the overall management responsibilities. Serverless architecture is another technology that is rapidly gaining popularity today.

This article will dig into PaaS and serverless. We'll discuss these two technologies, how they relate to each other, and how to utilize both in a modern development process.

Platform as a Service (PaaS) overview

Platform as a Service (PaaS) relates to services offered by a cloud platform that provides computing and software resources with minimal to no infrastructure management requirements. PaaS is the natural evolution of Infrastructure as Service (IaaS).

With IaaS, users have the freedom to spin up any number of resources like virtual servers, storage, and network infrastructure. But, they are also responsible for managing the underlying operating system, software configurations, etc.

In a PaaS offering, on the other hand, the cloud service provider manages the OS, underlying servers, network infrastructure, and most software configurations, leaving users free to develop and deploy applications rapidly.

(Read our comprehensive PaaS vs IaaS vs SaaS primer.)

Let's consider a simple web application deployment. There, we get a virtual server with an IaaS offering like AWS EC2. However, before we can deploy the application, we need to first:

1. Set up the web server.
2. Install dependencies plus other required software.
3. Set up network and storage.

However, with a PaaS offering like AWS Elastic Beanstalk, AWS will take care of EC2 instances, OS, web server installation, and any other resource configurations. Users simply spin up a beanstalk environment with the necessary configurations: all you need to do is provide proper configurations and deploy the application.

Apart from core server, storage, and networking services, PaaS also provides middleware, BI services, database systems, development tools. This alleviates the need for managing software licenses as they are handled by the cloud provider.

All these things are offered under a pay-as-you-go model that helps to cut off expenditure.

Advantages of PaaS

- Simplified and cost-effective resource management
- Ability to create readily scalable and highly available environments easily
- Reduction of infrastructure management and monitoring requirements, leading to time and cost savings
- Support for automation which reduces workload in the software development lifecycle
- Increased security
- More flexible development and deployment pipeline

Popular PaaS offerings

- AWS Elastic Beanstalk
- Azure App Service
- Azure Cognitive Search
- Google App Engine
- RedHat OpenShift
- IBM Cloud Pak for Applications
- BitNami by VMWare

Serverless overview

Serverless offerings aim to eliminate any management and configuration requirements from the software development and delivery process. With a serverless solution, users can write individual functions or services and deploy them directly in a serverless platform—without the need for any infrastructure or software configuration.

The cloud provider manages all the aspects of managing functions or services, such as scaling, availability needs, and managing bandwidth. Serverless is based on a usage-based monetization model where users only need to pay for the usage of the service.

The serverless approach enables developers to focus more on creating functions which leads to a more simplified development and deployment experience.

Serverless offerings provide the best solutions for microservices-based architectures or event-driven architecture.

For example, think of a web function that will be used to monitor a data stream. In traditional development, we need to:

1. Provision and configure the resources.
2. Set up the webserver.
3. Deploy the function with ongoing maintenance and cost (even when the function is not in use).

However, with a serverless solution like AWS Lambda, users only need to:

1. Select the programming language.
2. Create the function and frequency of the function execution or the triggering event.

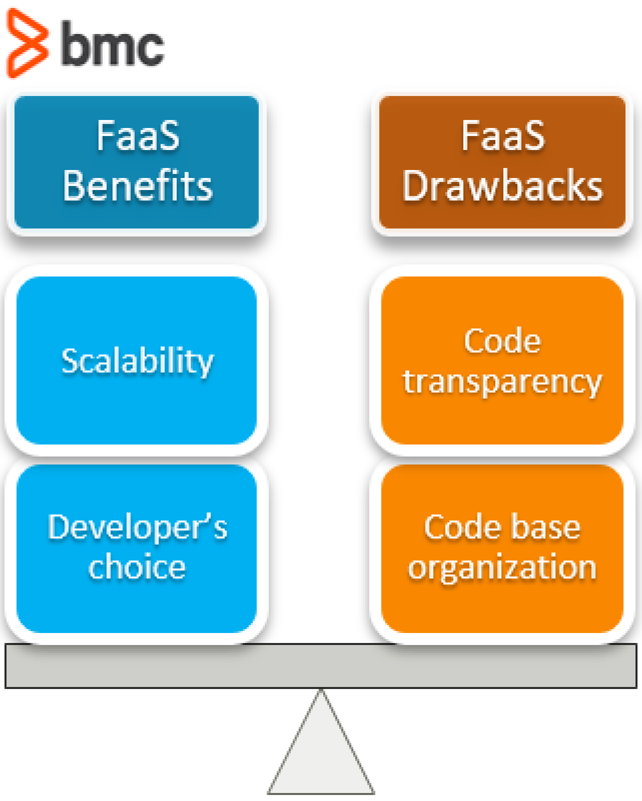
Then, the function will be automatically executed for the configured parameters. The function will be fully managed by the cloud provider and you'll only be billed for the number of executions.

The two main serverless offerings are FaaS and BaaS—let's briefly look at each.

Function as a Service (FaaS)

FaaS is the most widely used serverless solution. It allows users to execute code in stateless containers that are executed according to a predefined event. This provides developers a platform to easily develop and deploy code as functions without worrying about any infrastructure configurations.

(Learn more about FaaS.)



Backend as a Service (BaaS)

The BaaS model lets developers manage only the frontend and instead outsources most of the backend functionality to third-party services. BaaS provides services such as:

- Authentication
- Database management
- Hosting
- Notifications
- Etc.

In BaaS, an API or a software development kit (SDK) acts as the bridge between the user-managed frontend and the backend services. AWS Amplify, Azure Mobile Apps, and backendless are some available BaaS offerings.

Advantages of Serverless

- Eliminates any infrastructure configuration or management requirements
- Faster and simplified development and deployments due to the focus on individual functions or services
- Cost savings due to the usage-based payment model
- Highly scalable
- Reduced cloud provider lock-in

Popular serverless offerings

- AWS Lambda
- AWS Fargate

- Azure Functions
- Google Cloud Functions
- Apache OpenWhisk
- IBM Cloud Functions
- Fission.io serverless Kubernetes

Difference Between PaaS and Serverless

When comparing PaaS and serverless, the direct competitor will be FaaS offerings, as both provide a platform for developers to run their applications or functions. However, there are some significant differentiating factors between these two technologies, as mentioned below.

Scalability

A serverless application or a function will scale up and down automatically depending on the demand without any configuration requirement or intervention from the developer. A PaaS solution will also provide scalability, but the users will have to configure the scaling parameters—it does not automatically scale depending on the demand.

In general, the nature of serverless applications means they can scale up or down relatively faster than PaaS solutions.

Pricing structure

Serverless is based on a usage-based payment model where users only need to pay for the number of requests or execution time.

On the contrary, PaaS solutions are often based on pay-as-you-go models where a flat fee is charged for the service regardless of how much or little you use it. Users do have the ability to customize the resource requirements and manage costs in PaaS solutions, but they have no option to adjust the pricing dynamically based on the usage.

Still, compared to traditional deployments, each service provides cost savings. Ultimately the saving depends on the use case. For example, a highly active serverless function may lead to increased costs, which you can alleviate by moving to a PaaS solution with a predefined resource allocation that does not charge on the usage.

Control

PaaS offers more granular control over the underlying application and infrastructure configurations, unlike serverless, which offers no such control. However, this can be a pro or con depending on the requirement.

Since PaaS offers more control, it is easier to test and debug the application and understand the application logic (compared to serverless). Plus, you can use this to fine-tune the application at the application and infrastructure level. But beware—this control introduces increased complexity to the development and deployment process, which can be mitigated using serverless solutions.

Application architectures

Serverless applications are comparatively faster to startup and execute the functionality. However, the major drawback of serverless functions is their inability to properly support:

- Long-running processes
- Stateful applications

A long-running process will lead to cost constraints, while users will require separate mechanisms to manage states between each execution. This is why serverless is much more suited for event-driven architectures or microservices-based applications.

Vendor lock-in

Most organizations who use PaaS services are locked into that specific PaaS service with limited or no option to migrate to a different service provider. For example, an application that is deployed using AWS Elastic Beanstalk cannot be simply migrated to Azure App Service without complex restructuring.

On the other hand, you can easily migrate a serverless function between different service providers. That's because the function is bundled with the required dependencies and does not rely on any vendor-specific technologies. Users can simply:

1. Create a serverless function in a different provider with the same programming language.
2. Deploy the function and start using it immediately.

(See how to avoid vendor lock-in.)

So, is serverless the new PaaS?

Now we've got a clearer picture of both technologies, we can ask: is serverless the new PaaS? There is no clear-cut answer for this. Certainly, there are some overlaps between the functionality and use cases of each technology while they have distinct advantages and disadvantages over each other.

What solution or technology to choose solely depends on the user requirements, application architecture, and developer preferences. With all the leading cloud providers offering both options, it's wise for both organizations and users to select the option that best suits their needs.

Related reading

- BMC DevOps Blog
- Google Cloud Functions: An Introduction
- AWS Serverless Applications: The Beginner's Guide
- The Role of Microservices in DevOps
- Implementing GitOps To Deliver Cloud Native Applications