1. INTRODUCTION

Serverless computing is a cloud-computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity.[1] It can be a form of utility computing.

Serverless computing can simplify the process of deploying code into production. Scaling, capacity planning and maintenance operations may be hidden from the developer or operator. Serverless code can be used in conjunction with code deployed in traditional styles, such as microservices. Alternatively, applications can be written to be purely serverless and use no provisioned servers at all.This should not be confused with computing or networking models that do not require an actual server to function, like peer-to-peer (P2P).

The term *serverless computing* is a misnomer as the technology uses servers, only they're maintained by the provider. Serverless computing is considered a new generation of Platform as a Service, as in this case, the cloud service provider takes care of receiving client requests and responding to them, monitoring operations, scheduling tasks, and planning capacity. Thus, developers have no need to think about server and infrastructure issues and can entirely concentrate on writing software code.

Serverless computing is also sometimes called **Function as a Service** or event-based programming, as it uses functions as the deployment unit. The event-driven approach means that no resources are used when no functions are executed or an application doesn't run. It also means that developers don't have to pay for idle time. In addition, a serverless architecture ensures auto-scaling, allowing applications to provide users with services in spite of increasing workload. Thanks to all these features, serverless computing is considered a cost-saving, resource-limited, and fault-tolerant approach to software development. This novel approach is already provided by such industry giants as Amazon, Google, IBM, and Microsoft.

2. EVOLUTION OF SERVERLESS COMPUTING

(2.1) In the Beginning, There Was Hardware

Early on, evolution in enterprise architecture followed developments in hardware capabilities. Von Neumann architecture brought computer operating systems and software to the enterprise. In this model, the OS was tightly coupled to the hardware. But as operating systems moved to disk-based storage systems, it became possible to upgrade the OS without changing the hardware.

The first step away from a pure dependence on hardware had been made. But while operating systems were less dependent on specific hardware, software applications were conversely becoming more dependent on the specific OS. As operating systems became more complex, any given application would often have a hard dependency on a specific version of the OS or some other piece of software installed along with it.

(2.2) The Dawn of Virtualization

These dependencies created challenges in software development. Any new application would need to be tested to ensure that it worked in a range of specific environments. To this end, disk cloning tools were developed, which made it possible to make portable copies of those environments. This drew a clear distinction between operating systems and applications on one hand and the hardware required to run them on the other.

Hardware virtualization was perfected and then later commoditized by companies such as VMware. It became cheap and easy to create virtual machines. With the arrival of cloud computing, virtualization became a standard way to operate remote hardware owned and maintained by vendors such as Amazon Web Services. But virtual disk images were large and clumsy, creating significant scaling challenges.

(2.3) The Coming of Containers

Container technology, popularized by Docker, offered a much more lightweight alternative to images. Across any range of images, much of the software stack is likely to

be identical. In the Docker approach, only the differences from image to image need to be stored separately and the common elements can be drawn from an underlying container. This enabled microservice architecture, which promises unprecedented scalability in software development.

It turns out, though, that microservices aren't the be-all-and-end-all of scalability in enterprise architecture. There is still a lot of dead weight replicated across containers—specifically, libraries and boilerplate code. The code that really distinguishes one microservice from the next is just a small amount of application logic and nearly everything else is identical. More importantly, those nearly identical portions add little or no business value to the end customer.

(2.4) Thinking Outside the Container

From this realization sprouted new constructs—high-level "as-a-service" offerings such as Backend-as-a-Service (BaaS) and Function-as-a-Service (FaaS). These represented the dawn of serverless computing. The exact formulation of what is "serverless" is still up for grabs, but—broadly—we can define this as any application or service for which the enterprise provides only application logic, which is executed on infrastructure it does not manage.

The point is, key paradigms behind enterprise architecture and the applications it supports are shifting fast, presenting new opportunities for scalability and agility. Adopting the new paradigms will inevitably be a challenge in the context of monolithic legacy architectures and entrenched IT practices. But with serverless computing in place, you could greatly enhance your organization's ability to react to the inevitable but unforeseeable changes still to come.

Who Should Use Serverless?

You should especially consider using a serverless provider if you have a small number of functions that you need hosted. If your application is more complex, a serverless architecture can still be beneficial, but you will need to architect your application very

differently. This may not be feasible if you have an existing application. It may make more sense to migrate small pieces of the application into serverless functions over time.

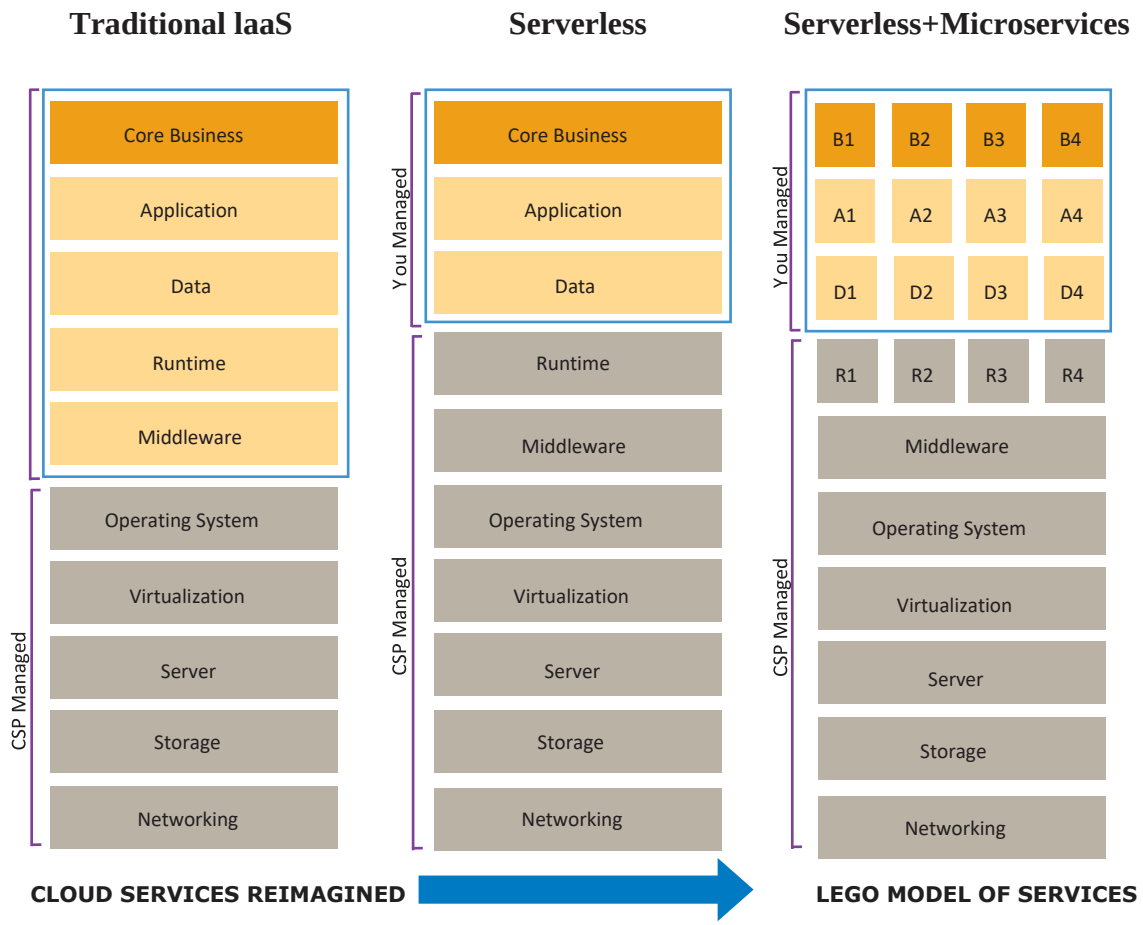
Using a product like Twilio Functions is especially helpful for builders and developers wanting to implement a Twilio solution. They are able to select pre-defined templates and deploy common communications use-cases without the need to host a server or use any other providers.

3. SERVERLESS ARCHITECTURE

(3.1) Serverless architecture characteristics

The solution is the serverless architecture. With this approach, you don't have to purchase, rent, or provision servers or virtual machines to run your code .Serverless architectures are denoted by these features:

- Built using services (not servers or virtual machines) provided by CSP and fully managed by the CSP
- Services that support elasticity and fault tolerance while providing enterprise-level global security n Automatic scaling n Built-in high availability n Integrated security
- Event-driven compute to execute business logic (examples include AWS Lambda, Azure functions, IBM Bluemix OpenWhisk, Google Cloud Functions n Pay-as-you go fee structure (for consumed services only) n Developer-productivity centric
- Promotes continuous build, integration, and deployment efforts
- Innovation by focusing on developing and deploying business functionality & Reduced time to market



(3.2) Use cases for serverless

When should you choose a serverless architecture over a server-based one? There are no specific rules, but examining the following attributes and use case possibilities can help you come to the right decision:

- Data in terms of variety (structured vs unstructured), volume, and variability
- Security requirements: Legal, regulatory, compliance,encryption, tokenization n Management control: Governance and reporting n Licensing: Third-party products, license extensibility
- Portability/Lock-ins: Ability to bring software components in house to port to different CSP, lock-ins
- Geographical restrictions: Serverless components may not be available in all geographies

- Integrations: Number and complexity of integrations with on-premise systems, other cloud services, and SaaS
- Feasibility of redesigning components and time to market n Resource availability and skill sets
- Cost of hosting infrastructure in cloud versus cost of running business function Serverless architectures can be useful in several situations:
- Web/Mobile applications n Streaming and telematics solutions n IoT solutions n Advanced analytics and arti□cial intelligence solutions n Cloud automation and DevOps enablement

  Before making an architectural decision, it's critical to make a thorough evaluation of the attributes and use cases.

Serverless architecture run applications that depend on external Function as a Service and Backend as a Service providers, which run the application code in temporary containers.

Consequently, a serverless architecture includes the following three core components:

- **API gateway** — This is a fully managed service that's used to define, deploy, and maintain APIs. For API integrations, developers use standard HTTPS requests that connect their presentation tier with the functions they write on a serverless platform. Some serverless providers like Amazon offer their own API gateways for their clients.
- **Function as a Service (FaaS)** — This is the layer that executes the application code (business logic) through multi-protocol triggers. Function instances are provisioned on request (in a few milliseconds) in compliance with associated configurations.
- **Backend as a Service (BaaS)** — This is a cloud computing service model that serves as a cloud-based distributed database, eliminating the need for administrative overhead. BaaS features include cloud storage, push notifications, server code, user and file management, and many other backend services. These services have their own APIs so developers can easily integrate them into their applications.

(3.3) HOW APPLICATIONS RUN IN SERVERLESS ENVIRONMENT

To create an application, developers should consider how to break it into smaller functions to reach a granular level for serverless compatibility. Each function needs to be written in one of the programming languages supported by the serverless provider.

Function deployment

Deploying a function is considerably different from deploying a traditional system as there's no server on the developer's side. In serverless computing, a developer just uploads function code to the serverless platform and the provider is responsible for its execution as well as for provisioning and allocating resources, maintaining the virtual machine, and managing processes. The platform vendor also takes care of automatic horizontal and elastic scaling.

FaaS functions are executable on a container with few resources. Functions are triggered through requests and events when necessary without requiring the application to be running all the time. Requests that can trigger FaaS functions are specified by the provider. For instance, such triggers can be inbound HTTP requests, scheduled tasks, S3 updates, or messages added to a message bus. Functions can also be invoked via a platform-provided API, externally, or within the cloud environment.

Function limitations

Functions in serverless computing have significant architectural restrictions related to their state and duration of execution. FaaS functions are stateless, which means that they have no machine state. Thus, external databases are necessary to simulate state behavior of functions.

Moreover, functions are limited in their execution time, which determines how long each invocation can be run. This restriction makes developers create several short-running functions instead of one long-running task.

In order to initialize an instance of a function in response to an event, a FaaS platform requires some time for a cold start. This leads to startup latency that depends on various factors and has an unpredictable duration. It may be caused by the programming language, the number of libraries you use, the configuration of the function environment, and so on. Though latency raises many concerns, the latency your app experiences will depend on the traffic and type of your application. If a function repeats frequently, a cold start will be rare, as the platform reuses the instance of a function and its host container from a previous event.

4. BENEFITS OF SERVERLESS COMPUTING

Serverless computing is more beneficial than existing cloud services as it offers better application performance along with reduced operational costs. Let's consider other benefits of cloud computing that attract so much attention to this technology.

- **Reduced cloud costs —** Serverless computing is based on the principle of pay-as-you-go. It costs nothing when your application doesn't run. Developers pay only for the time when their application executes a user's functions in response to specific events or requests. This model greatly benefits developers, as they can significantly save cloud costs.
- **Reduced development costs** — With serverless computing, there's no need to handle updates or infrastructure maintenance as developers can now rent most of the resources necessary for software development.
- **Improved resource management** — Applications developed in serverless environments are more fine-grained. This means that the cloud provider can closely match abstract demand to actual system resources. When an application doesn't run, the cloud provider distributes the server resources among other running applications. Once a triggering event appears, resources are allocated to execute the function.

- **Elastic scalability** — A serverless architecture has the ability to scale up and down according to application workload. This is achieved by replicating functions. Developers no longer need to purchase additional infrastructure for handling unexpected growth.
- **Fewer responsibilities for developers** — In the serverless model, the cloud provider has more control over resources and more insight into the context of application behavior. Developers don't need to care about workload intensity, resource distribution, scaling, and application deployment as these issues are in the hands of the cloud provider.
- **Faster releases and reduced time to market** — To deploy new functions, developers just need to compile their code, zip it, and upload it to the serverless platform. There's no need to write any scripts to deploy functions. This leads to faster releases and time-to-market reductions of up to two-thirds according to a study by Microsoft.
- **Agile-friendly development** — Serverless computing allows developers to concentrate on application code rather than infrastructure maintenance. Moreover, it benefits developers through reduced software complexity and better code optimization. For instance, if an application usually takes one second to execute an operation with a hardware server, in a serverless environment it may take only 200 milliseconds, so developers can save 80 percent of their costs.
- **Multi-language support** — Currently, some serverless platforms support multiple programming languages, so developers can choose the most convenient for them.
- **Built-in logging and monitoring mechanisms** — Serverless providers have developed their own solutions for user logging and monitoring that eliminate the need for developers to purchase third-party tools for similar purposes. In addition, serverless providers offer function-level auditing that ensures the application data privacy that's necessary for full GDPR compliance.

5. CHALLENGES OF SERVERLESS COMPUTING

Despite the many benefits, you should understand that serverless computing is a relatively new technology that also has certain challenges.

- **Costs of serverless services —** While serverless computing can significantly reduce your development costs, this isn't always true with different cloud computation approaches. For instance, serverless functions are currently most attractive for CPU bound computations, while I/O bound functions are not as affordable as on dedicated virtual machines or containers.
- **Cold start** — The key benefit of serverless computing is its ability to scale to zero so that developers don't need pay for idle time. However, this may lead to increased invocation latency due to cold starts. For instance, it may occasionally take up to ten full seconds to run the first invocation of a JVM-implemented function. Preemptive warming up could be a good solution to this problem, but developers may have to pay penalties for getting their serverless code ready to run. Another solution could be using tools that can predict system load and analyze the duration of load spikes.
- **Resource limits —** Resource limits are necessary to ensure that the platform can deal with spikes and withstand attacks. Serverless computing imposes limits on memory, function execution time, bandwidth, and CPU usage. For instance, the maximum execution time per request is 300 seconds (or 5 minutes) for Azure Functions and AWS Lambda.
- **Inadequate application testing —** Though it's easy to test different functions of an application, it may be challenging to test the infrastructure and the combination of all functions. The reason for this is the complexity of the serverless architecture, which makes it difficult to manage the countless endpoints in different environments.
- **Increased security concerns** — There are high security risks in a serverless environment, as many users run their functions on a shared platform. Thus, if someone loads malicious code, it may negatively affect all cloud users.

- **Vendor compatibility** — FaaS providers operate on different platforms and support different programming languages. Thus, software changes may be inevitable if developers want to change providers.
- **Monitoring and debugging —** Since developers have no control over the provider's servers, there are limited opportunities for identifying problems and bottlenecks. After a function is executed, it only leaves traces in logs recorded by the serverless platform. In addition, there's a lack of debugging tools that can operate in serverless environments. This is because debugging distributed systems requires access to a substantial number of relevant metrics in order to identify the root causes of problems.
- **Function state —** While real applications require state, it's not clear how to manage state in serverless functions, as they're stateless. In order to overcome this challenge, programming models, libraries, and tools need to provide the necessary levels of abstraction.
- **Execution time limitations —** Serverless functions are limited in their execution time, but not all applications can run only with short-running functions. There are cases when long-running logic is necessary, for instance for financial or emergency applications.
- **Availability of skills** — Serverless computing is a fairly new business, so there's a lack of developers on the market with skills in serverless programming. Software companies may have to invest in training their developers in order to catch up with new business trends.
- **Composability** — Some functions are designed to call other functions and coordinate the execution of several functions that may result in parallel execution. To overcome this challenge, it's necessary to use tools that can facilitate the creation and maintenance of compositions.
- **Multi-tenancy concerns** — In addition to your application, a serverless provider also supports other clients. Thus, there's a risk that your software will be placed on the server along with function-heavy applications that may cause slow performance.

- **Access to the file system level** — A serverless architecture isn't the best choice for software that requires access to the file system or operating system level. These types of applications need to do things such as read attributes from configuration files or split in-memory cache to disk, but serverless functions don't allow for this.

(5.1) How To Overcome Challenges

Serverless is growing — not just the technology or the adoption, but also what we mean by the term "serverless" itself. Instead of just Functions-as-a-Service (most famously AWS Lambda), we can understand serverless today as applying to all fully-scalable event-driven infrastructures where you don't manage the server. A typical serverless application will include multiple services and functions, all connected together — services like databases (DynamoDB), file storage solutions (S3), messaging services (SQS) and API triggers (API Gateway), often connected together by serverless functions. Together, they jointly create an end-to-end application.

This is a step beyond the containerized microservices that have become familiar in recent years. Serverless apps are effectively built from multiple "nanoservices" that each performs a single, specialized role, within a specific microservice.

Fundamentally, as Simon Wardley argues, serverless changes computing into a commodity like electricity. When electricity first became widely available, new companies went with the cheaper, lighter, easier-to-use electric machines. Older businesses, though, already had their kerosine-powered behemoth production lines and were much slower to switch. Serverless faces a similar adoption pattern, though at a much faster rate. New software companies are becoming the first to take advantage of the flexibility, scalability and cost savings. Older, more-established enterprises are starting to experiment with serverless as an extension to their existing monolith or microservices based applications, but wide use of this technology might take more time.

Serverless is not an all or nothing methodology. Teams don't need to replace their entire technology stack; they can start with some small components connecting to their legacy and, as time goes by, transfer more and more workloads when it makes sense.

We're noticing that this is a familiar adoption pattern for serverless: a software engineer in a company will start playing around with serverless functions for fun, and maybe connect them to Slack or Alexa to provide some specific added functionality. From there, it's a short step to using Lambdas for automating some basic IT Operations or monitoring tasks like using them to check on resource usage once an hour or to take database snapshots. The next step is usually starting to use Lambdas in a business-oriented application, like for data processing in an ETL pipeline.

Some early innovators like CapitalOne, iRobot, Netflix, AirBnB are using serverless extensively. But other companies will incrementally move towards there, adding more serverless components as they go.

Amazon, the biggest cloud provider, announced loads of new serverless products and features in November's Re:Invent, and other cloud providers are all making their own advances in widening the serverless product space.

There are still some limitations that need more work. The "cold start" issue for functions and services is a concern for many serverless adopters (perhaps a misplaced concern in some cases, but that's another story!) Function concurrency limits can be another problem when you start to scale.

Logging is another concern. One serverless application uses multiple components, services, programming languages, regions, and sometimes even several cloud providers. Individual services can create logs, but following the application, flow is a challenge because there's no simple way of tracking a request from one service or function to another. If there's a problem, it can be hard to pinpoint the root cause failure: did the function time out because of a bug in the function? Or maybe elsewhere in one of the distributed components? Was the data malformed somewhere upstream in the request flow?

There's a need for the right tools to provide the needed visibility and observability to ensure quick and simple troubleshooting and optimization.

More broadly, there are two big challenges that are holding back the adoption of this new and emerging technology. The first is the lack of knowledge by the engineering teams and the second is the lack of supporting tools in the technology ecosystem, to help developers with everyday tasks. The first challenge can be addressed by the cloud vendors sharing knowledge and advice through developer advocates, and by the developer community sharing its knowledge and experience through publications, blogs, and community events.

The second challenge is addressed by many startups that build the needed tools to support the new technology. Companies like Serverless Inc. PureSec, Lumigo, Dashbird and more are filling this gap.

We are in a transition period, where software organizations are experimenting in the serverless waters. The best way to adopt new technology is a phased approach, starting with a small scale project and growing from there. Managing this transition likely to be a growth industry for software architects and specialist consultancy firms who can help companies manage the change.

Cloud computing is growing dramatically. Engineering methodologies should evolve to support cloud native applications, with the emphasis on increasing development velocity. Serverless methodology does exactly that, moving non-critical tasks to the cloud provider and focusing team efforts on the business logic to move faster. This is the promise of serverless, and we're seeing it begin to be fulfilled.

6. SERVERLESS PROVIDERS IN PUBLIC CLOUD

The term *serverless* became popular in 2014 after Amazon launched AWS Lambda, a serverless platform that was followed by the Amazon API Gateway in 2015. By the middle of 2016, the *serverless* concept had given birth to the Serverless Conference, after which other vendors began to introduce their own solutions to the market. Currently, all tech giants provide a serverless architecture.

- **Amazon's Lambda** is the most mature serverless platform on the market. It supports a variety of programming languages including Java, Python, and Node.js. Lambda takes advantage of most AWS services and allows developers to apply its functions as event handlers as well as to provide glue code when composing services.
- **IBM Cloud Functions** is another serverless deployment that acts as an event action platform for creating composite functions. It supports Node.js, Java, Swift, Python, and arbitrary binaries embedded in a Docker container. Cloud Function is based on OpenWhisk, which is available on GitHub under an Apache open source license.
- **Microsoft Azure Functions** lets developers write code for functions like processing images, sending emails, and maintaining files that are executed on schedule or when needed. The platform provides HTTP webhooks, integration with Azure, and support for such programming languages as .NET, Go, Node.js, Python, PHP, and Java. The runtime code is open-source and is available on GitHub under an MIT License.
- **Google Cloud Functions** is a computing platform that provides basic FaaS functionality to run serverless functions written in Node.js in response to HTTP calls and events from Google Cloud services. It provides development tools such as prediction APIs, translation APIs, and data storage.

In addition to these serverless providers, there are several other serverless projects including OpenLambda, Webtask, LeverOS, Pivotal Cloud Foundry 2.0, Spotinst Functions, and Gestalt Framework.

7. SERVERLESS RUNTIMES

Most, but not all, serverless vendors offer compute runtimes, also known as function as a service (FaaS) platforms, which execute application logic but do not store data. The first "pay as you go" code execution platform was Zimki, released in 2006, but it was not commercially successful.In 2008, Google released Google App Engine, which featured metered billing for applications that used a custom Python framework, but could not execute arbitrary code. PiCloud, released in 2010, offered FaaS support for Python.

AWS Lambda, introduced by Amazon in 2014,was the first public cloud infrastructure vendor with an abstract serverless computing offering.

Google Cloud Platform offers Google Cloud Functions since 2016.

IBM offers IBM Cloud Functions in the public IBM Cloud since 2016.

Microsoft Azure offers Azure Functions, offered both in the Azure public cloud or on-premises via Azure Stack.

Oracle introduced Fn Project, an open source serverless computing framework offered on Oracle Cloud Platform and available on GitHub for deployment on other platforms.

In addition, there are a number of open source serverless projects with various levels of popularity and usage:

OpenWhisk was initially developed by IBM with contributions from RedHat, Adobe, and others. OpenWhisk is the core technology in IBM Cloud Functions.

Project Riff is an open source serverless platform implementation built on Kubernetes by Pivotal Software. Project Riff is the foundation of Pivotal Function Service.

8. SERVERLESS DATABASES

Several serverless databases have emerged in the last few years. These systems extend the serverless execution model to the RDBMS, eliminating the need to provision or scale virtualized or physical database hardware.

Amazon Aurora offers a serverless version of its databases, based on MySQL and PostgreSQL, providing on-demand, auto-scaling configurations.

Azure Data Lake is a highly scalable data storage and analytics service. The service is hosted in Azure, Microsoft's public cloud. Azure Data Lake Analytics provides a distributed infrastructure that can dynamically allocate or de-allocate resources so customers pay for only the services they use.

Google Cloud Datastore is an eventually-consistent document store. It offers the database component of Google App Engine as a standalone service. Firebase, also owned by Google, includes a hierarchical database and is available via fixed and pay-as-you-go plans.

FaunaDB is a strongly consistent, globally distributed serverless database that offers a GraphQL API. It can be accessed directly via a function-as-a-service provider, or from Javascript in the browser. In FaunaDB, data is replicated across multiple datacenters running on Amazon Web Services, Google Cloud Platform and Microsoft Azure.

9. CLOUD v/s SERVERLESS COMPUTING

(9.1) WHAT IS CLOUD COMPUTING ?

Cloud computing is a buzzword that's been used and abused to the point where it might mean very different things to different people. To find some consensus, let's see what the authorities on the matter have to say about it.

From Amazon:

"Cloud computing is the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet with pay-as-you-go pricing."

And here's a definition from Microsoft:

"Simply put, cloud computing is the delivery of computing services—servers, storage, databases, networking, software, analytics, intelligence and more—over the Internet ("the cloud") to offer faster innovation, flexible resources, and economies of scale. You typically pay only for cloud services you use, helping lower your operating costs, run your infrastructure more efficiently, and scale as your business needs change."

To boil it down to the essentials: computer services are delivered over the internet and you pay for what you use. It's efficient and it scales. Simple enough.

Here's where things become more tricky…

(9.1.1) TYPES OF CLOUD COMPUTING

According to Microsoft, there are 4 major types of cloud computing:

1. **Infrastructure as a Service (IaaS)**
2. **Platform as a Service (PaaS)**
3. **Software as a Service (SaaS)**
4. **Serverless**

To stay focused on the question at hand, we won't bother getting into the definitions of these categories of cloud computing. But, to further your understanding of what Serverless is and how it relates to other forms of cloud computing, it's worth noting that these four cloud computing categories build on top of each other, so they're commonly referred to as the "cloud computing stack."

(9.2) WHAT IS SERVERLESS COMPUTING?

"Serverless" is a bit of a misnomer or, at least, a somewhat misleading label, because somewhere in the world in enormous warehouses, there are definitely actual servers powering "Serverless" computing.

But, the term points to what ultimately makes Serverless so innovative and valuable: *with Serverless you don't have to think about the server at all.* So, *in effect*, it's "serverless," though not technically or literally, and that's where the term comes from. "The setup, capacity planning, and server management are invisible to you because they're handled by the cloud provider," according to Microsoft, and, "Serverless applications don't require you to provision, scale, and manage any servers," according to Amazon.

(9.2.1) THE SERVER DISAPPEARS (FROM YOUR LIST OF CONCERNS)

Traditionally, if you were a developer you'd need to do quite a bit of work to set up and maintain a server for a new app or website. This process can be frustrating and time-consuming if you're not a sysadmin by trade. Worse, if you make a mistake and configure something incorrectly it can lead to serious consequences — like a security breach, downtime, or an inefficient use of resources (you end up paying more than you have to for hosting).

And, that's exactly why Serverless is so appealing — it's all taken care of. The chore of provisioning a server is done for you and you're relieved of the ongoing burden of making sure everything is secure, up-to-date, and optimized. All server-related concerns are taken off your plate and handled by the Serverless cloud architecture.

(9.2.2) HOSTING RESOURCES ARE SUPPLIED ON DEMAND

To get into the more technical side of what is happening under the hood, the functions of the site/app are split out into separate containers and resources are applied to specific functions as needed. Whatever your site/app needs to run itself is served up on a silver platter. When your application needs more memory, it gets more memory allocated in real-time. When your application receives a thousand web requests, it gives you the compute cycles and bandwidth to deliver those requests. Your application has a need for computer resources, and the Serverless architecture ensures it has exactly what it needs when it needs it. And, this is another defining characteristic of Serverless…

(9.2.3) PRECISION ALLOCATION OF RESOURCES

Serverless delivers exact units of resources in response to a demand from the application. Contrast that with traditional cloud computing where chunks of resources need to be allocated in advance so that they're available when they're needed.

With traditional cloud hosting, you might add 2GB or 4GB of RAM so that your application has sufficient memory available for peaks in usage. With Serverless, your application might request and be allocated exactly 3.76GB of RAM to complete some task. The allocation is exactly what is needed to meet the demand of the app/site.

With traditional cloud computing, the computer resources are dedicated to you whether you're using them or not while with Serverless, you're dynamically pulling *only what you need* from a vast ocean of resources.

(9.2.4) PAY ONLY FOR WHAT YOU USE

While the common definition of cloud computing speaks about "only paying for what you use," Serverless delivers on that promise more literally. Sure, when you use Infrastructure as a Service you're only paying for the server resources you asked for, but you're still paying for 8GB of memory whether your application is using all 8GB at this moment or not.

With Serverless, you pay only for the exact amount of resources it takes to perform a function. If your website is only consuming 3.39 GB of memory at this moment, that's all you'll pay for. For most websites there will be a constant flux in resource usage as a result of fluctuating demand. Serverless automatically adapts to those fluctuations (a term called elasticity) so that you literally only pay for what your application uses moment to moment.

This means that your Serverless hosting bill will vary month to month depending on what your website uses. If you have a slow month, your bill might be very low. If next month your traffic explodes, your bill will be much higher. In this way, Serverless

computing is quite efficient. There is very little waste because your hosting plan automatically adjusts up and down. The downside is that this fluctuation can make billing unpredictable which can make it difficult to forecast a budget. Organizations that place a lot of value on predictable budgets may be better suited to more traditional hosting options, like VPS plans.

SUMMARY

Serverless hosting offers some unique benefits over common cloud computing that make it an attractive option for many businesses:

- No need to manage or interact with a server
- Computing resources are supplied as needed to scale a site automatically
- Resources are allocated precisely rather than in chunks
- You pay only for the resources that are consumed

10. NEED OF SERVERLESS COMPUTING

Serverless enables you to build modern applications with increased agility and lower total cost of ownership. Building serverless applications means that your developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on-premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great products which scale and that are reliable.

11. FUTURE OF SERVERLESS

We're coming to the end of this journey into the world of Serverless architectures. To close out I'm going to discuss a few areas where I think the Serverless world may develop in the coming months and years.

Mitigating the drawbacks

Serverless is still a fairly new world. As such, the previous section on drawbacks was extensive, and I didn't even cover everything I could have. The most important developments of Serverless are going to be to mitigate the inherent drawbacks and remove, or at least improve, the implementation drawbacks.

Tooling

Tooling continues to be a concern with Serverless, and that's because so many of the technologies and techniques are new. Deployment/application bundling and configuration have both improved over the last two years, with the Serverless framework and Amazon's Serverless Application Model leading the way. However the "first 10 minutes" experience still isn't as universally amazing as it could be, although Amazon and Google could look to Microsoft and Auth0 for more inspiration.

An area I've been excited to see being actively addressed by cloud vendors is higher-level release approaches. In traditional systems, teams have typically needed to code their own processes to handle "traffic-shifting" ideas like blue-green deployment and canary releases. With this in mind Amazon supports automatic traffic shiftingfor both Lambda and API Gateway. Such concepts are even more useful in Serverless systems where so many individually deployed components make up a system—atomic release of 100 Lambda functions at a time is simply not possible. In fact, Nat Pryce described to me the idea for a "mixing desk" approach, one where we can gradually bring groups of components in and out of a traffic flow.

Distributed monitoring is probably the area in need of the most significant improvement. We've seen the early days of work here from Amazon's X-Ray and various third-party products, but this is definitely not a solved problem.

Remote debugging is also something I'd like to see more widespread. Microsoft Azure Functions supports this, but Lambda does not. Being able to breakpoint a remotely running function is a very powerful capability.

Finally, I expect to see improvements for tooling of "meta operations"—how to more effectively look after hundreds or thousands of FaaS functions, configured services, etc. For instance, organizations need to be able to see when certain service instances are no longer used (for security purposes, if nothing else), they need better grouping and visibility of cross-service costs (especially for autonomous teams that have cost responsibilities), and more.

State management

The lack of persistent in-server state for FaaS is fine for a good number of applications, but it's a deal breaker for many others—whether it be for large cache sets or fast access to session state.

One workaround for high-throughput applications will likely be for vendors to keep function instances alive for longer between events, and let regular in-process caching approaches do their job. This won't work 100 percent of the time since the cache won't be warm for every event, but this is the same concern that already exists for traditionally deployed apps using auto-scaling.

A better solution could be very low-latency access to out-of-process data, like being able to query a Redis database with very low network overhead. This doesn't seem too much of a stretch given that Amazon already offer a hosted Redis solution in their Elasticache product, and that they already allow relative co-location of EC2 (server) instances using Placement Groups.

More likely, though, I think we're going to see different kinds of hybrid (Serverless and non-Serverless) application architectures embraced to take account of the externalized-state constraint. For instance, for low-latency applications you may see an approach of a regular, long-running server handling an initial request, gathering all the context necessary to process that request from its local and external state, then handing off a fully contextualized request to a farm of FaaS functions that don't need to look up data externally.

Platform improvements

Certain drawbacks to Serverless FaaS right now come down to the way platforms are implemented. Execution duration, startup latency, and cross-function limits are three obvious ones. These will likely either be fixed by new solutions or given workarounds with possible extra costs. For instance, I imagine that startup latency could be mitigated by allowing a customer to request that two instances of a FaaS function are always available at low latency, with the customer paying for this availability. Microsoft Azure Functions has elements of this idea with Durable Functions, and App Service plan-hosted functions.

Of course we'll see platform improvements beyond just fixing current deficiencies, and these will be exciting too.

Education

Many vendor-specific inherent drawbacks with Serverless are being mitigated through education. Everyone using such platforms needs to think actively about what it means to have so much of their ecosystems hosted by one or many application vendors. We need to think about questions like, "Do we want to consider parallel solutions from different vendors in case one becomes unavailable?" and "How do applications gracefully degrade in the case of a partial outage?"

Another area for education is technical operations. Many teams now have fewer sysadmins than they used to, and Serverless is going to accelerate this change. But sysadmins do more than just configure Unix boxes and Chef scripts—they're often the people on the front line of support, networking, security, and the like.

A true DevOps culture becomes even more important in a Serverless world since those other non-sysadmin activities still need to get done, and often it's developers who are now responsible for them. These activities may not come naturally to many developers and technical leads, so education and close collaboration with operations folk is of utmost importance.

Increased transparency and clearer expectations from vendors

Finally, on the subject of mitigation: vendors are going to have to be even more clear in the expectations we can have of their platforms as we rely on them for more of our hosting capabilities. While migrating platforms is hard, it's not impossible, and untrustworthy vendors will see their customers taking their business elsewhere.

CONCLUSION

Adopting serverless can deliver many benefits—but the road to serverless can get challenging depending on the use case. And like any new technology innovations, serverless architectures will evolve en route to becoming a well-established obvious standard. While serverless architecture may not be a solution to every IT problem, it surely represents the future of many kinds of computing solutions in the coming years

Serverless computing is a novel and promising approach to software development. The technology eliminates the complexity of dealing with servers and reduces development costs. It lets developers concentrate on development and stop worrying about budget limitations thanks to pay-per-use billing. Though serverless computing is a new technology full of challenges, the Apriorit team has already used it in our projects. If you're looking for cost-effective yet high-quality and time-efficient software development, we can offer you our services.

BIBLIOGRAPHY

- **Web References :**

aws.amazon.com / Serverless computing

https://en.wikipedia.org Serverless Computing – Wikipedia

https://www.cloudflare.com

https://azure.microsoft.com

https://martinfowler.com

www.techopedia.com (Serverless Computing Definition)

https://cloud.google.com

https://www.twilio.com

https://www.computerworld.com

https://www.techbeacon.com (Serverless computing & Architecture)