DOI:10.1145/3368454

The server is dead, long live the server.

BY PAUL CASTRO, VATCHE ISHAKIAN,
VINOD MUTHUSAMY, AND ALEKSANDER SLOMINSKI

The Rise of Serverless Computing

CLOUD COMPUTING IN general, and Infrastructure-as-a-Service (IaaS) in particular, have become widely accepted and adopted paradigms for computing with the offerings of virtual machines (VM) on demand. By 2020, 67% of enterprise IT infrastructure and software spending will be for cloud-based offerings.[16]

A major factor in the increased adoption of the cloud by enterprise IT was its pay-as-you-go model where a customer pays only for resources leased from the cloud provider and have the ability to get as many resources as needed with no up-front cost (elasticity).[2] Unfortunately, the burden of scaling was left for developers and system designers that typically used overprovisioning techniques to handle sudden surges in service requests. Studies of reported usage of cloud resources in datacenters[19] show a substantial gap between the resources that cloud customers allocate and pay for (leasing VMs), and actual resource utilization (CPU, memory, and so on).

Serverless computing is emerging as a new and compelling paradigm for the deployment of cloud applications, largely due to the recent shift of enterprise application architectures to containers and microservices.[23] Using serverless gives pay-as-you-go without additional work to start and stop server and is closer to original expectations for cloud computing to be treated like as a utility.[2] Developers using serverless computing can get cost savings and scalability without needing to havea high level of cloud computing expertise that is time-consuming to acquire.

Due to its simplicity and economical advantages, serverless computing is gaining popularity as reported by the increasing rate of the "serverless" search term by Google Trends. Its market size is estimated to grow to 7.72 billion by 2021.[10] Most prominent cloud providers including Amazon, IBM, Microsoft, Google, and others have already released serverless computing capabilities with several additional open source efforts driven by both industry and academic institutions (for example, see CNCF Serverless Cloud Native Landscape[a]).

From the perspective of an IaaS customer, the serverless paradigm shift presents both an opportunity and a risk. On the one hand, it provides developers with a simplified programming model for creating cloud applications that abstracts away most, if

a https://s.cncf.io/

» key insights

- **Serverless computing takes the original promises of cloud computing and delivers true pay only for resources used with almost infinite scalability while hiding the details of how servers are used and maintained.**
- **Serverless computing is a new cloud computing paradigm with enormous economic growth potential.**
- **Serverless computing allows the developer to focus on developing business logic and gives the cloud provider additional control over optimizing resources.**
- **There are many technical challenges and opportunities for research.**

ILLUSTRATION BY PETER BOLLINGER

RIGHTSLINK

not all, operational concerns. They no longer have to worry about availability, scalability, fault tolerance, over/underprovisioning of VM resources, managing servers, and other infrastructure issues. Instead, they can focus on the business aspects of their applications. The paradigm also lowers the cost of deploying cloud code by charging for execution time rather than resource allocation. On the other hand, deploying such applications in a serverless platform is challenging and requires relinquishing design decisions to the platform provider that concern, among other things, quality-of-service (QoS) monitoring, scaling, and fault-tolerance properties. There is a risk an application's requirements may evolve to conflict with the capabilities of the platform.

From the perspective of a cloud provider, serverless computing is an additional opportunity to control the entire development stack, reduce operational costs by efficient optimization and management of cloud resources, offer a platform that encourages the use of additional services in their ecosystem, and lower the effort required to author and manage cloud-scale applications.

One of the major challenges slowing the adoption of serverless is the lack of tools and frameworks.

Defining Serverless Computing

Serverless computing can be defined by its name—less thinking (or caring) about servers. Developers do not need to worry about low-level details of servers management and scaling, and only pay for when processing requests or events. We define serverless as follows:

Serverless computing is a platform that hides server usage from developers and runs code on-demand automatically scaled and billed only for the time the code is running.

This definition captures the two key features of serverless computing:

▸ *Cost—billed only for what is running (pay-as-you-go).* As servers and their usage is not part of serverless computing model, then it is natural to pay only when code is running and not for idle servers. As execution time may be short, then it should be charged in fine-grained time units (like hundreds of milliseconds) and developers do not need to pay for overhead of servers creation or destructions (such as VM booting time). This cost model is very attractive to workloads that must run occasionally; serverless essentially supports "scaling to zero" and avoid need to pay for idle servers. The big challenge for cloud providers is the need to schedule and optimize cloud resources.

▸ *Elasticity—scaling from zero to "infinity."* Since developers do not have control over servers that run their code, nor do they know the number of servers their code runs on, decisions about scaling are left to cloud providers. Developers do not need to write auto-scaling policies or define how machine-level usage (CPU, memory, and so on) translates to application usage. Instead they depend on the cloud provider to automatically start more parallel executions when there is more demand for it. Developers also can assume the cloud provider will take care of maintenance, security updates, availability and reliability monitoring of servers.

Serverless computing today typically favors small, self-contained units of computation to make it easier to manage and scale in the cloud. A computation, which can be interrupted or restarted, cannot depend on the cloud platform to maintain its state. This inherently influences the serverless computing programming models. There is, however, no equivalent notion of scaling to zero when it comes to state, since a persistent storage layer is needed. However, even if the implementation of a stateful service requires persistent storage, a provider can offer a pay-as-you-go pricing model that would make state management serverless. We are seeing providers releasing services that stretch the definition of serverless, and the definition may evolve over time. For example, Amazon Aurora is a "serverless" database service, which supports powerful auto-scaling capabilities but requires minimum memory and CPU allocations and hence does not scale to zero and has ongoing costs.[b]

The most natural way to use serverless computing is to provide a piece of code (function) to be executed by the serverless computing platform. It leads to the rise of Function-as-a-service (FaaS) platforms focused on

b https://aws.amazon.com/rds/aurora/serverless/

allowing small pieces of code represented as functions to run for limited amount of time (at most minutes), with executions triggered by events or HTTP requests (or other triggers), and not allowed to keep persistent state (as function may be restarted at any time). By limiting time of execution and not allowing functions to keep persistent state FaaS platforms can be easily maintained and scaled by service providers. Cloud providers can allocate servers to run code as needed and can stop servers after functions finish as they run for limited amount of time. If functions must maintain state, then they can use external services to persist their state.

FaaS is an embodiment of serverless computing principles, which we define as follows:

Function-as-a-Service is a serverless computing platform where the unit of computation is a function that is executed in response to triggers such as events or HTTP requests.

Our approach to defining serverless is consistent with emerging definitions of serverless from industry. For example, Cloud Native Computing Foundation (CNCF) defines serverless computing[11] as "the concept of building and running applications that do not require server management. It describes a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and then executed, scaled, and billed in response to the exact demand needed at the moment." While our definition is close to the CNCF definition, we make a distinction between serverless computing and providing functions as a unit of computation. As we discuss in the research challenges section, it is possible that serverless computing will expand to include additional aspects that go beyond today's relatively restrictive stateless functions into possibly long-running and stateful execution of larger compute units. However, today serverless and FaaS are often used interchangeably as they are close in meaning and FaaS is the most popular type of serverless computing.

Paul Johnston (co-founder of ServerlessDays) defined serverless as follows:[c] "A serverless solution is one that costs you nothing to run if nobody is using it (excluding data storage)." This definition highlights the most important characteristic of serverless computing—pays-as-you-go. It assumes serverless computing is a subset of cloud computing so auto-scaling is included and developers have no access to servers. CNCF and our definitions emphasize not only pay-as-you-go or "scale to zero" aspects, but also the lack of need to manage servers.

Another way to define serverless computing is by what functionality it enables. Such an approach emphasizes "serverless is really about the managed services" and FaaS can be treated as cloud "glue," as described by Steven Faulkner (a senior software engineer at LinkedIn[d]). It is "glue" that joins applications composed of cloud services. Such a definition addresses only a narrow set of use cases where serverless computing is used, while our definition captures the important use cases, which we will highlight in the accompanying sidebars.

All definitions share the observation that the name 'serverless computing' does not mean servers are not used, but merely that developers can leave most operational concerns of managing servers and other resources, including provisioning, monitoring, maintenance, scalability, and fault-tolerance to the cloud provider.

History and Related Work

The term 'serverless' can be traced to its original meaning of not using servers and typically referred to peer-to-peer (P2P) software or client-side only solutions.[28] In the cloud context, the current serverless landscape was introduced during an AWS re:Invent event in 2014.[3] Since then, multiple cloud providers, industrial, and academic institutions have introduced their own serverless platforms. Serverless seems to be the natural progression following recent advancements and adoption of VM and container technologies, where each step up the abstraction layers led to more lightweight units of computation in terms of resource consumption, cost, and speed of development and deployment. Furthermore, serverless builds upon long-running trends and advances in both distributed systems, publish-subscribe systems, and event-driven programming models,[12] including actor models,[1] reactive programming,[4] and active database systems.[25]

Serverless platforms can be considered an evolution of Platform-as-a-Service (PaaS) as provided by platforms such as Cloud Foundry, Heroku, and Google App Engine (GAE). PaaS was defined by NIST[e] as "the capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations." In this definition, users are expected to manage deployments of applications and have control over hosting environment configurations.

Serverless FaaS, when compared to this definition of PaaS, is removing user control over hosting to provide simpler scaling and more attractive billing model: the cloud provider controls the hosting environment's configuration, runs user-provided code only when it is invoked, and only bills for actual usage while hiding the complexity of scaling (in practice implementing auto-scaling in PaaS is not easy and it is very difficult to scale to zero). That is a significant change when compared to the previous generation of PaaS (which could be considered first generation of PaaS) and it is very attractive for PaaS users that do not need to pay for idle resources and avoid managing auto-scaling rules.

The main differentiators of serverless platforms is transparent autoscaling and fine-grained resource charging only when code is running. That should not to be confused with free-usage quota, where limited monthly resource quota is available, but counted even if the application is not used. For example, GAE Standard is priced in "instance hours"[f] and even if the app is

c http://bit.ly/2G3Hp1R
d http://bit.ly/2xzNEWB
e http://bit.ly/2lXCgkI
f http://bit.ly/2kuqZbh

RIGHTSLINK

Table 1. Comparison of different choices for cloud as a service.

	IaaS	1st Gen PaaS	FaaS	BaaS/SaaS
Expertise required	High	Medium	Low	Low
Developer Control/Customization allowed	High	Medium	Low	Very low
Scaling/Cost	Requires high-level of expertise to build auto-scaling rules and tune them	Requires high-level of expertise to build auto-scaling rules and tune them	Auto-scaling to work load requested (function calls), and only paying for when running (scale to zero)	Hidden from users, limits set based on pricing and QoS
Unit of work deployed	Low-level infrastructure building blocks (VMs, network, storage)	Packaged code that is deployed and running as a service	One function execution	App-specific extensions
Granularity of billing	Medium to large granularity: minutes to hours per resource to years for discount pricing	Medium to large granularity: minutes to hours per resource to years for discount pricing	Very low granularity: hundreds of milliseconds of function execution time	Large: typically, subscription available based on maximum number of users and billed in months

not used the instance is kept running. Later, GAE added Flexible version with a more fine-grained billing unit, but still developers will be billed even if server is not used. That can lead to unexpected outcomes when the bill arrives at the end of month for forgotten test services.[g]

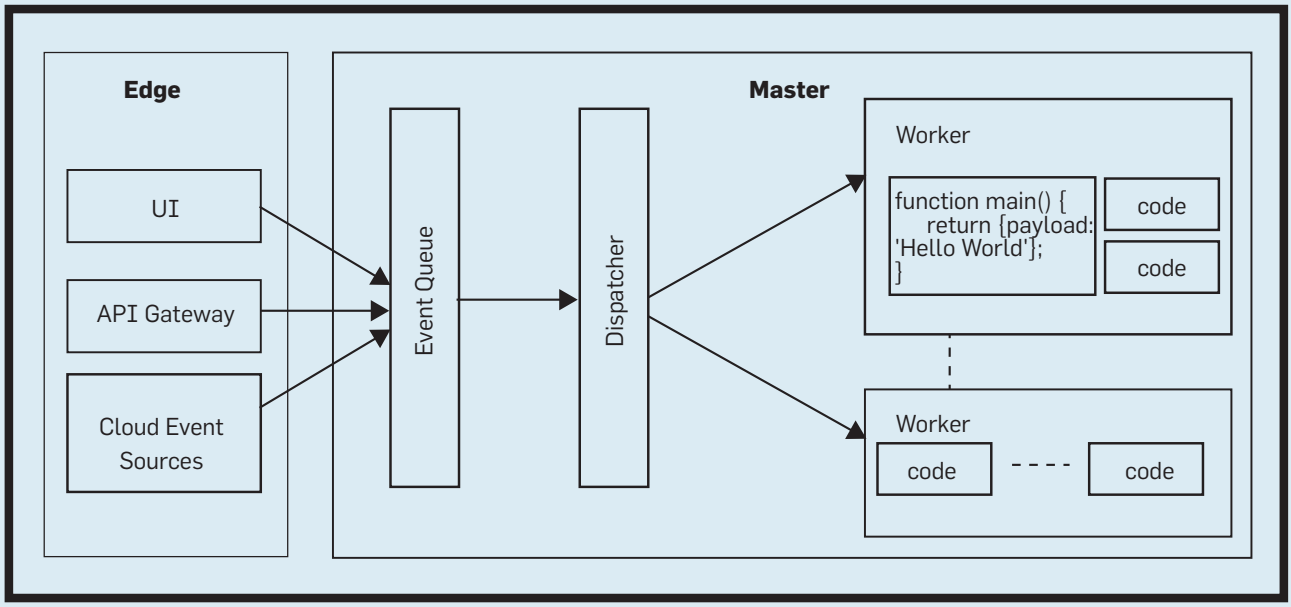
Mobile Backend as-a-Service (MBaaS) or more generalized Backend as-a-Service (BaaS) bears a close resemblance to serverless computing. Some of those services even provided "cloud functions" (for example, Facebook's now-defunct Parse Cloud Code). Such code, however, was typically limited to mobile use cases.

Software-as-a-Service (SaaS) may support the server-side execution of user-provided functions, but they are executing in the context of an application and hence limited to the application domain. Some SaaS vendors allow the integration of arbitrary code hosted somewhere else and invoked via an API call. For example, this is approach is used by the Google Apps Marketplace in Google Apps for Work.

The boundaries defining serverless computing functionality overlaps with PaaS and SaaS. One way to categorize serverless is to consider the varying levels of developer control over the infrastructure. In an IaaS model, the developer has much more control over the resources, but is responsible for managing both the application code and operating the infrastructure. This gives the developer great flexibility and the ability to customize every aspect of the application and infrastructure, such as administering VMs, managing capacity and utilization, sizing the workloads, achieving fault tolerance and high availability. PaaS abstracts away VMs and takes care of managing underlying operating systems and capacity, but the developer is responsible for the full life cycle of the code that is de-

g https://stackoverflow.com/questions/47125661/

Figure 1. High-level serverless FaaS platform architecture.



ployed and run by the platform, which does not scale down to zero. SaaS represent the other end of the spectrum where the developer has no control over the infrastructure, and instead get access to prepackaged components. The developer is allowed to host code there, though that code may be tightly coupled to the platform. BaaS is similiar to SaaS in that the functionality is targeting specific use cases and components, for example, MBaaS provide backend functionality needed for mobile development such as managing push notifications, and when it allows developer to run code it is within that backend functionality (see Table 1).

Architecture

The core functionality of a FaaS framework is simply that of an event processing system, as shown in Figure 1. The service manages a set of user defined functions (a.k.a actions). Once a request is received over HTTP from an event data source (a.k.a. triggers), the system determines which action(s) should handle the event, create a new container instance, send the event to the function instance, wait for a response, gather execution logs, make the response available to the user, and stop the function when it is no longer needed.

The abstraction level provided by FaaS is unique: a short-running stateless function. This has proven to be both expressive enough to build useful applications, but simple enough to allow the platform to autoscale in an application agnostic manner.

While the architecture is relatively simple, the challenge is to implement such functionality while considering metrics such as cost, scalability, latency, and fault tolerance. To isolate the execution of functions from different users in a multitenant environment, container technologies,[9] such as Docker, are often used.

Upon the arrival of an event, the platform proceeds to validate the event ensuring it has the appropriate authentication and authorization to execute. It also checks the resource limits for that particular event. Once the event passes validation, the platform the event is queued to be processed. A worker fetches the request, allocates the appropriate container, copies over

Figure 2. A function written in JavaScript.

```
function main(params, context) {
    return {payload: 'Hello, ' + params.name
                    + ' from ' + params.place};
}
```

Table 2. Real-world applications that use serverless computing.

Where is serverless used?	What do they use serverless computing for?
Aegex	Xamarin application that customers can use to monitor real-time sensor data from IoT devices.[a]
Abilisense	Manages an IoT messaging platform for people with hearing difficulties. They estimated they could handle all the monthly load for less than $15 a month.[b]
A Cloud Guru	Uses functions to perform protected actions such as payment processing and triggering group emails. In 2017 they had around 200K users and estimated $0.14 to deliver video course to a user.[c]
Coca-Cola	Serverless Framework is a core component of The Coca-Cola Company's initiative to reduce IT operational costs and deploy services faster.[d] One particular use case is the use of serverless in their vending machine and loyalty program, which managed to have 65% cost savings at 30 million hits per month.[e]
Expedia	Expedia did "over 2.3 billion Lambda calls per month" back in December 2016. That number jumped 4.5 times year-over-year in 2017 (to 6.2 billion requests) and continues to rise in 2018.[f] Example applications include integration of events for their CI/CD platforms, infrastructure governance and autoscaling.[g]
Glucon	Serverless mobile backend to reduce client app code size and avoid disruptions.[h]
Heavywater Inc	Runs Website and training courses using serverless (majority of cost per user is not serverless but storage of video). Serverless reduced their costs by 70%.[i]
iRobot	Backend for iRobot products.[j]
Postlight	Mercury Web Parser is a new API from Postlight Labs that extracts meaningful content from Web pages. Serving 39 million requests for $370/month, or: How We Reduced Our Hosting Costs by Two Orders of Magnitude.[k]
PyWren	Map-reduce style framework for highly parallel analytics workloads.[l]
WeatherGods	A mobile weather app that uses serverless as backend.[m]
Santander Bank	Electronic check processing. Less than $2 to process all paper checks within a year.[n]
Financial Engines	Mathematical calculations for evaluation and optimization of investment portfolios. 94% savings on cost approximately 110K annually.[o]

a http://bit.ly/2XCas2J
b https://thenewstack.io/ibms-openwhisk-serverless/
c https://gotochgo.com/2017/sessions/61
d http://bit.ly/2xCWYsO
e https://aws.amazon.com/serverless/videos/video-lambda-coca-cola/
f http://bit.ly/2JktVAR
g https://www.youtube.com/watch?v=gT9x9LnU_rE
h http://bit.ly/2lyLaoOs
i http://bit.ly/2xzYE6t
j https://aws.amazon.com/solutions/case-studies/irobot/
k http://bit.ly/2YQcUni
l http://pywren.io/
m https://thenewstack.io/ibms-openwhisk-serverless/
n http://bit.ly/2YHtsxY
o https://aws.amazon.com/solutions/case-studies/financial-engines/

RIGHTSLINK

Use Case 1: Event Processing

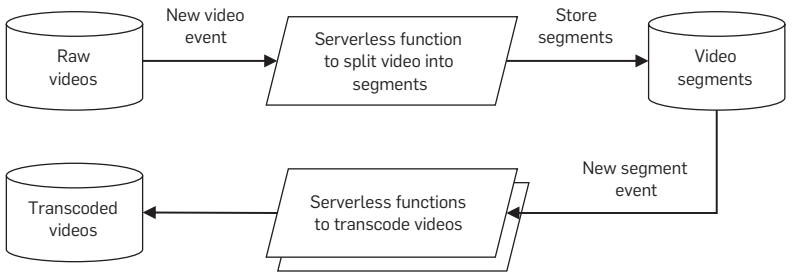
One class of applications that exemplify the use of serverless is event-based programming. The following use case shows an example of a bursty, compute-intensive workload popularized by AWS Lambda, and has become the "Hello, World" of serverless computing. It is a simple image-processing event handler function.

Netflix uses serverless functions to process video files (https://amzn.to/2xMtwAt). The videos are uploaded to Amazon S3,[23] which emits events that trigger Lambda functions that split the video and transcode them in parallel to different formats. The flow is depicted in the figure here.

The function is completely stateless and idempotent, which has the advantage that in the case of failure (such as network problems accessing the S3 folder), the function can be executed again with no side effects.

While the example here is relatively simple, by combining serverless functions with other services from the cloud provider, more complex applications can be developed, for example, stream processing, filtering and transforming data on the fly, chatbots, and Web applications.

Video processing.



the function—use code from storage use—into the container and executes the event. The platform also manages stopping and deallocating resources for idle function instances.

Creating, instantiating, and destroying a new container for each function invocation while can be expensive and introduces an overall latency, which is referred to as the *cold start problem*. In contrast, warm containers are containers that were already instantiated and executed a function. Cold start problems can be mitigated by techniques such as maintaining a pool of uninstantiated stem cell containers, which are containers that have been previously instantiated but not assigned to a particular user, or reuse a warm container that have been previously invoked for the same user.[7] Another factor that can affect the latency is the reliance of the user function on particular libraries (for example, numpy) that must be downloaded and installed before function invocation. To reduce startup time of cloud functions, one can appropriately cache the most important packages across the node workers, thus leading to reduced startup times.[24]

In typical serverless cloud offerings, the only resource configuration customers are allowed to configure is the size of main memory allocated to a function. The system will allocate other computational resources (for example, CPU) in proportion to the main memory size. The larger the size, the higher the CPU allocation. Resource usage is measured and billed in small increments (for example, 100ms) and users pay only for the time and resources used when their functions are running.

Several open source serverless computing frameworks are available from both industry and academia (for example, Kubeless, OpenLambda, OpenWhisk, OpenFaaS). In addition, major cloud vendors such as Amazon, IBM, Google, and Microsoft have publically available commercial serverless computing frameworks for their consumers. While the general properties (for example, memory, concurrent invocations, maximum execution duration of a request) of these platforms are relatively the same, the limits as set by each cloud provider are different. Note the limits on these properties are a moving target and are constantly changing as new features and optimizations are adopted by cloud providers. Evaluating the performance of different serverless platforms to identify the trade-offs has been a recent topic of investigation,[17,20,26] and recent benchmarks have been developed to compare the serverless offering by the different cloud providers.[h]

Programming Model

A typical FaaS programming model consists of two major primitives: *Action* and *Trigger*. An Action is a stateless function that executes arbitrary code. Actions can be invoked asynchronously in which the invoker—caller request—does not expect a response, or synchronously where the invoker expects a response as a result of the action execution. A Trigger is a class of events from a variety of sources. Actions can be invoked directly via REST API, or executed based on a trigger. An event can also trigger multiple functions (parallel invocations), or the result of an action could also be a trigger of another function (sequential invocations). Some serverless frameworks provide higher-level programming abstractions for developers, such as function packaging, sequencing, and composition, which may make it easier to construct more complex serverless apps.

Currently, serverless frameworks execute a single main function that takes a dictionary (such as a JSON object) as input and produces a dictionary as output. They have limited expressiveness as they are built to scale. To maximize scaling, serverless functions do not maintain state between executions. Instead, the developer can write code in the function to retrieve and update any needed state. The function is also able to access a context object that represents the environment in which the function is running (such as a security context). As shown in Figure 2, a func-

h http://faasmark.com/

tion written in JavaScript could take as input a JSON object as the first parameter, and context as the second.

Current Cloud Provider Serverless offerings support a wide variety programming languages, including Java, Python, Swift, C#, and Node.js. Some of the platforms also support extensibility mechanisms for code written in any language as long as it is packaged in a Docker image that supports a well-defined API.

Due to the limited and stateless nature of serverless functions, and its suitability for composition of APIs, cloud providers are offering an ecosystem of value added services that support the different functionalities a developer may require, and is essential for production ready applications. For example, a function may need to retrieve state from permanent storage, such as a file server or database, another may use a machine learning service to perform some text analysis or image recognition. While the functions themselves may scale due to the serverless guarantees, the underlying storage system itself must provide reliability and QoS guarantees to ensure smooth operation.

Tools and Frameworks

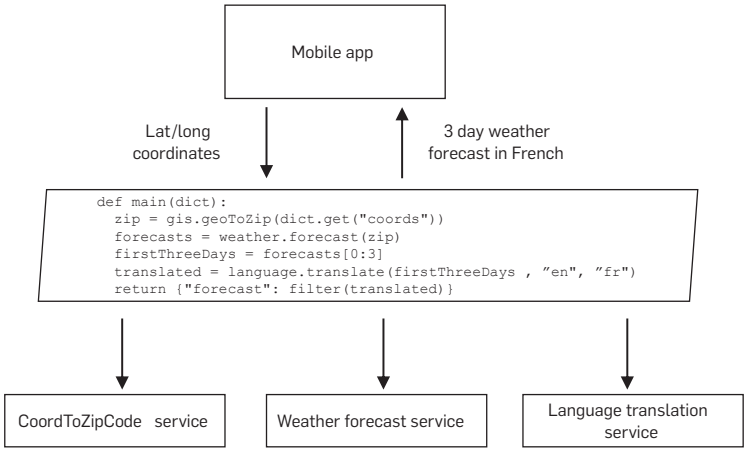
One of the major challenges slowing the adoption of serverless is the lack of tools and frameworks. The tools and frameworks currently available can be categorized as follows: development, testing, debugging, deployment. Several solutions been proposed to deal with these categories.

Almost all cloud providers provide a cloud-based IDE, or extensions/plugins to popular IDEs that allows the developer to code and deploy serverless functions. They also provide a local containerized environment with an SDK that allows the developer to develop and test locally serverless functions before deploying it in a cloud setting. To enable debugging, function execution logs are available to the developer and recent tools such as AWS X-Ray[i] allow developers to detect potential causes of the problem.[22] Finally, there are open source frameworks[j] that allow developers to define serverless functions, triggers, and services needed by the functions. Theses frameworks will handle the deployment of these functions to the cloud provider.

Use Case 2: API Composition

Consider a mobile app that sequentially invokes a geolocation, weather, and language translation APIs to render the weather forecast for a user's current location. A short serverless function can be used to invoke these APIs. Thus the mobile app avoids invoking multiple APIs over a potentially resource constrained mobile network connection, and offloads the filtering and aggregation logic to the backend. Glucon, for example, used serverless in its conference scheduler application to minimize client code, and avoid disruptions.

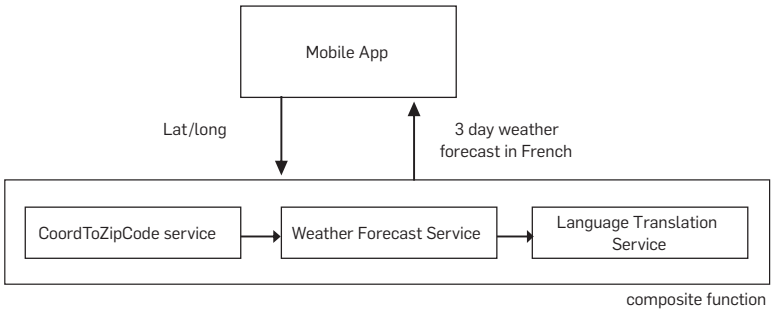
A serverless anti-pattern of offloading API calls from mobile app to backend.



Note the main function is acting as an orchestrator that is waiting for a response from a function before invoking another, thus incurring a cost of execution while the function is basically waiting for I/O. Such a pattern of programming is referred to as a serverless anti-pattern.

The serverless programming approach would be to encapsulate each API call as serverless function, and the chain the invocation of these functions in a sequence. The sequence itself behaves as a composite function.

Offloading API calls and glue logic from mobile app to backend.



More complex orchestrations can use technologies like AWS Step Functions and IBM Composer to prevent serverless anti-patterns but may incur additional costs due to the services.

Use Cases

Serverless computing has been utilized to support a wide range of applications. From an infrastructure perspective, serverless and more traditional architectures may be used interchangeably or in combination. The determination of when to use serverless will likely be influenced by other non-functional requirements, such as the amount of control over operations

i https://aws.amazon.com/xray/
j https://serverless.com/

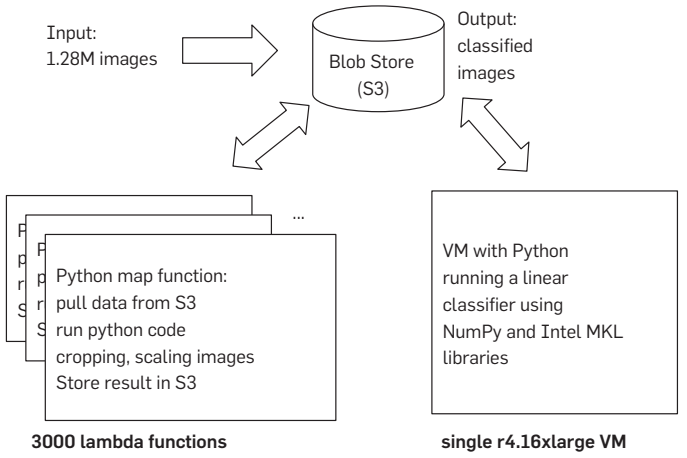
RIGHTSLINK

Use Case 3: Map-Reduce Style Analytics

PyWren[18] (illustrated in the figure) is a Python-based system that utilizes the serverless framework to help users avoid the significant development and management overhead of running MapReduce jobs. It is able to get up to 40TFLOPS peak performance from AWS Lambda, using AWS S3 for storage and caching. A similar reference architecture has been proposed by AWS Labs (https://github.com/awslabs/lambda-refarch-mapreduce).

PyWren exemplifies a class of use cases that uses a serverless platform for highly parallel analytics workloads.

Map + monolithic Reduce PyWren example implementing ImageNet Large Scale Visual Recognition Challenge.



required, cost, as well as application workload characteristics.

From a cost perspective, the benefits of a serverless architecture are most apparent for bursty[5,27] workloads. Bursty workloads fare well because the developer offloads the elasticity of the function to the platform, and just as important, the function can scale to zero, so there is no cost to the consumer when the system is idle.

There are many areas where serverless computing is used today. Table 2 provides a representative list of different types of applications used in different domains along with a short description. We emphasize this list is not exhaustive; we offer it to identify and discuss emerging patterns. Interested readers can find examples by going through additional use cases that are publically available by cloud providers.

From a programming model perspective, the stateless nature of serverless functions lends themselves to application structure similar to those found in functional reactive programming. This includes applications that exhibit event-driven and flow-like processing patterns (see Use Case 1 sidebar of thumbnail creation).

As a comparison, consider an equivalent solution implemented as an application running on a set of provisioned VMs. The logic in the application to generate the thumbnails is relatively straightforward, but the user must manage the VMs, including monitoring traffic loads, autoscaling the application, and managing failures. There is also a limit to how quickly VMs can be added in response to bursty workloads, forcing the user to forecast workload patterns and pay for pre-provisioned resources. The consequence is there will always be idle resources, and it is impossible to scale down to zero VMs. In addition, there must be a component that monitors for changes to the S3 folder, and dispatch these change events to one of the application instances. This dispatcher itself must be fault-tolerant and auto-scale.

Another class of applications that exemplify the use of serverless is composition of a number of APIs, controlling the flow of data between two services, or simplify client-side code that interacts by aggregating API calls (see Use Case 2 sidebar).

Serverless computing may also turn out to be useful for scientific computing. Having ability to run functions and not worry about scaling and paying only for what is used can be very good for computational experiments. One class of applications that started gaining momentum are compute intensive applications.[13] Early results show (see Use Case 3 sidebar) the performance achieved is close to specialized optimized solutions and can be done in an environment that scientists prefer such as Python.

If the workloads cannot be easily divided into smaller units (such as Python functions), then batch-oriented systems such as high-performance computing (HPC) or MapReduce clusters are a better option. If the demand for such clusters can be sustained, for example, by having job queues where jobs are submitted and scheduled based on available resources, then workloads can be executed more cheaply, albeit possibly taking longer to complete. The cost is lower than using FaaS as the service provider can get cheaper VMs either by buying actual servers, using vendor platforms such as Databricks or BigQuery, or getting reserved VMs with longer contracts. If batch workloads can tolerate occasional restarts it may be better to run such workloads with on-demand VMs (such as AWS spot instances).

Many "born in cloud" companies build their services to take full advantage of cloud services. Whenever possible they use existing cloud services and built their functionality using serverless computing. Before serverless computing they would need to use virtual machines and create autoscaling policies. Serverless computing, with its ability to scale to zero and almost infinite on-demand scalability, allows them to focus on putting business functionality in serverless functions instead of becoming experts in low-level cloud infrastructure and server management (see Use Case 4 sidebar for more details).

RIGHTSLINK

Challenges and Limitations

Serverless computing is a large step forward, and is receiving a lot of attention from industry and is starting to gain traction among academics. Changes are happening rapidly and we expect to see different evolutions of what is serverless and FaaS. While there are many immediate innovation needs for serverless,[6,14,15] there are significant challenges that need to be addressed to realize full potential to serverless computing. Based on discussions during a series serverless workshops organized by the authors (https://www.serverlesscomputing.org/workshops/), and several academic[21] and industrial surveys (https://www.digitalocean.com/currents/june-2018/), we outline the following challenges:

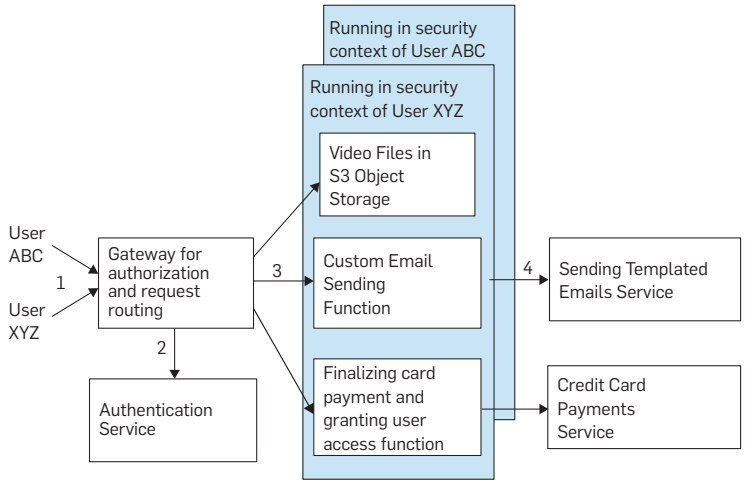
Programming models and tooling: since serverless functions are running for shorter amounts of time there will be multiple orders of magnitude more of them that compose applications (for example, SparqTV; http://bit.ly/2xFktSb), a video-streaming service runs more than 150 serverless functions). This however, will make it more difficult to debug and identify bottlenecks. Traditional tools that assumed access to servers (for example, root privilege) to monitor and debug applications are not applicable for serverless applications, and new approaches are needed. Although some of these tools are starting to become available, higher-level development IDEs, and tools for orchestrating and composing applications, will be critical. In addition, the platform may need to be extended with different recovery semantics, such as at-least-once or at-most-once, or more sophisticated concurrency semantics, such as atomicity where function executions are serialized. As well, refactoring functions (for example, splitting and merging them), and reverting to older versions, must be fully supported by the serverless platform. While these problems have received a lot of attention from industry and academia,[22] there is still a lot of progress to be made.

Lack of standards and vendor lock-in: Serverless computing and FaaS are new and quickly changing and currently there are no standards. As the area matures, standards can be expected to emerge. In the meantime, developers can use tools and frameworks that allow the use of different serverless computing providers interchangeably.

Use Case 4: Multi-Tenant Cloud Services

A Cloud Guru is a company whose mission is to provide users with cloud training that includes videos. An important part of their business model is providing service on-demand and optimizing delivery cost. Their usage patterns are unpredictable and may change depending on holidays or if they do promotions. They must be able to scale and to isolate users for security reasons while providing for each user backend functionality such as payment processing or sending email messages.

Requests are authenticated and routed to a custom function that runs in isolation and with the user's context.



They achieve this by leveraging cloud services and serverless computing to build a multi-tenant, secure, highly available, and scalable solution that can run each user specific code as serverless functions (http://bit.ly/2JyPfBK). This dramatically simplifies how a multi-tenant solution is architected as shown in the figure. A typical flow starts with a user making a request (1) from a frontend application (Web browser). The request is authenticated (2) by using an external service and then sent either to a cloud service (such as object store to provide video files) or (3) to a serverless function. The function makes necessary customizations and typically invokes other functions or (4) cloud services.

Research Opportunities

Since serverless is a new area, there are many opportunities for the research community to address. We highlight some options:

System-level research opportunities: A key differentiator of serverless is the ability to scale to zero, and not charge the customers for idle time. Scaling to zero, however, leads to problems of cold starts, particularly for functions with customized library requirements.[17] Techniques to minimize the cold start problem while still scaling to zero are critical. A more fundamental question currently being examined is if containers are the right abstractions for running serverless applications and whether abstractions with smaller footprints, such as unikernels, are more suitable.

Legacy code in serverless: Serverless application designs are fundamentally different from typical legacy applications. The economical value of existing code represents a huge investment of countless hours of developers coding and debugging software. One of the most important problems may be to what degree existing legacy code can be automatically or semiautomatically decomposed into smaller-granularity pieces to take advantage of these new economics.

RIGHTSLINK

Stateful serverless: Current serverless platforms are mostly stateless, and it is an open question if there will be inherently stateful serverless applications in the future with different degrees of QoS without sacrificing the scalability and fault-tolerance properties.

Service-level agreements (SLA): Serverless computing is poised to make developing services easier, but providing QoS guarantees remains difficult.[17,27] While the serverless platform needs to offer some guarantees of scalability, performance, and availability, this is of little use if the application relies on an ecosystem of services, such as identity providers, messaging queues, and data persistence, which are outside the control of the serverless platform. To provide certain QoS guarantees, the serverless platform must communicate the required QoS requirements to the dependent components. Furthermore, enforcement may be needed across functions and APIs, through the careful measurement of such services, either through a third-party evaluation system, or self-reporting, to identify the bottlenecks.

Serverless at the edge: There is a natural connection between serverless functions and edge computing as events are typically generated at the edge with the increased adoption of IoT and other mobile devices. iRobot's use of AWS Lambda and step functions for image recognition was described by Barga as an example of an inherently distributed serverless application.[8] Recently, Amazon extended its serverless capabilities to an edge based cloud environment by releasing AWS Greengrass. Consequently, the code running at the edge, and in the cloud may not just be embedded but virtualized to allow movement between devices and cloud. That may lead to specific requirements that redefine cost. For example, energy usage may be more important than speed.

New serverless applications: The serverless programming model is inherently different, but that should be a motivation to think about building—or rebuilding—new and innovative solutions that tap into what it can provide. Pywren,[18] ExCamera,[13] HPC, numerical analysis, and AI chatbots are but some examples of how developers are using serverless to come up with new solutions and applications.

Conclusion

Serverless computing is an evolution in cloud application development, exemplified by the Function-as-a-Service model where users write small functions, which are then managed by the cloud platform. This model has proven useful in a number of application scenarios ranging from event handlers with bursty invocation patterns, to compute-intensive big data analytics. Serverless computing lowers the bar for developers by delegating to the platform provider much of the operational complexity of monitoring and scaling large-scale applications. However, the developer now needs to work around limitations on the stateless nature of their functions, and understand how to map their application's SLAs to those of the serverless platform and other dependent services. While many challenges remain, there have been rapid advances in the tools and programming models offered by industry, academia, and open source projects. C

References

1. Agha, G. An overview of actor languages. In *Proceedings of the 1986 SIGPLAN Workshop on Object-Oriented Programming*, 58–67. ACM, New York, NY.
2. Armbrust, M. et al. A view of cloud computing. *Commun. ACM 53*, 4 (2010), 50–58; https://m.cacm.acm.org/magazines/2010/4/81493-a-view-of-cloud-computing/fulltext
3. AWS re:invent 2014—(mbl202) new launch: Getting started with AWS lambda; https://www.youtube.com/watch?v=UFj27laTWQA.
4. Bainomugisha, E., Carreton, A.L., . Cutsem, V., Mostinckx, S. and Meuter, W.D. A survey on reactive programming. *ACM Comput. Surv. 45*, 4 (Aug. 2013), 52:1–52:34.
5. Baldini, I., Castro, P., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Suter, P. Cloud-native, event-based programming for mobile applications. In *Proceedings of the Intern. Conf. on Mobile Software Engineering and Systems*, 2016, 287–288. ACM, New York, NY.
6. Baldini, I. et al. Serverless computing: Current trends and open problems. *Research Advances in Cloud Computing*, Springer, 2017, 1–20.
7. Baldini, I., Cheng, P., Fink, S.J., Mitchell, N., Muthusamy, V., Rabbah, R., Suter, P. and Tardieu, O. The serverless trilemma: Function composition for serverless computing. In *Proceedings of the 2017 ACM SIGPLAN Intern. Symp. on New Ideas, New Paradigms, and Reflections on Programming and Software*.
8. Barga, R.S. Serverless computing: Redefining the cloud [Internet]. In *Proceedings of the 1st Intern. Workshop on Serverless Computing* (Atlanta, GA, USA, June 5, 2017); http://www.serverlesscomputing.org/wosc17/#keynote
9. Bernstein, D. Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing 1*, 3 (Sept. 2014), 81–84.
10. Businesswire. $7.72 billion function-as-a-service market 2017—Global forecast to 2021: Increasing shift from DevOps to serverless computing to drive the overall Function-as-a-Service market; https://bwnews.pr/2G3ZzQY.
11. CNCF Serverless White Paper; https://github.com/cncf/wg-serverless#whitepaper
12. Etzioni, O. and Niblett, P. *Event Processing in Action*. Manning Publications Co., Greenwich, CT, 2010.
13. Fouladi, S., Wahby, R.S., Shacklett, B., Balasubramaniam, K., Zeng, W., Bhalerao, R., Sivaraman, A., Porter, G. and Winstein, K. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. *NSDI* (2017), 363–376
14. Fox, G.C., Ishakian, V., Muthusamy, V. and Slominski, A. Status of Serverless Computing and Function-as-a-Service (FaaS) in Industry and Research. Technical Report; arXiv:1708.08028, 2017
15. Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H. Serverless computation with openlambda. In *Proceeding of the 8th USENIX Workshop on Hot Topics in Cloud Computing* (Denver, CO, USA, June 20–21, 2016).
16. IDC. IDC FutureScape: Worldwide IT Industry 2017 Predictions. IDC #US41883016, 2016.
17. Ishakian, V., Muthusamy, V. and Slominski, A. Serving deep learning models in a serverless platform. In *Proceedings of the IEEE Intern. Conf. on Cloud Engineering*, 2018
18. Jonas, E., Pu, Q., Venkataraman, S., Stoica, I. and Recht, B. Occupy the cloud: Distributed computing for the 99%. In *Proceedings of the 2017 Symp. on Cloud Computing*.
19. Kilcioglu, C. Rao, J.M. Kannan, A. and McAfee, R.P. Usage patterns and the economics of the public cloud. In *Proceedings of the 26th Intern. Conf. World Wide Web*, 2017
20. Lee, H., Satyam, K. and Fox, G.C. Evaluation of production serverless computing environments. In *Proceedings of IEEE Cloud Conf. Workshop on Serverless Computing* (San Francisco, CA, 2018).
21. Leitner, P., Wittern, E., Spillner, J. and Hummer, W. A mixed-method empirical study of Function-as-a-Service software development in industrial practice; https://peerj.com/preprints/27005
22. Lin, W-T, Krintz, C., Wolski, R., Zhang, M., Cai, X., Li, T. and Xu, W. Tracking causal order in AWS lambda applications. In *Proceedings of the IEEE Intern. Conf. on Cloud Engineering*, 2018.
23. NGINX. NGINX announces results of 2016 future of application development and delivery survey; http://bit.ly/2YM27e2/.
24. Oakes, E., Yang, L., Houck, K., Harter, T., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H. Pipsqueak: Lean Lambdas with large libraries. In *Proceedings of 2017 IEEE 37th Intern. Conf. on Distributed Computing Systems Workshops*, 395–400.
25. Paton, N.W. and Díaz, O. Active database systems. *ACM Comput. Surv. 31*, 1 (1999), 63–103.
26. Wang, L., Li, M., Zhang, Y., Ristenpart, T. and Swift, M. Peeking behind the curtains of serverless platforms. In *Proceedings of USENIX Annual Technical Conf.*, 2018, 133–146. USENIX Association.
27. Yan, M., Castro, P., Cheng, P., Ishakian, V. Building a chatbot with serverless computing. In *Proceedings of the 1st Intern. Workshop on Mashups of Things*, 2016.
28. Ye, W., Khan, A.I. and Kendall, E.A. Distributed network file storage for a serverless (P2P) network. In *Proceedings of the 11th IEEE Intern. Conf. on Networks*, 2003, 343–347.

Paul Castro (castrop@us.ibm.com) is a research staff member at IBM T.J. Watson Research Center in Yorktown Heights, NY, USA.

Vatche Ishakian (vishakian@bentley.edu) is an assistant professor at Bentley University, Waltham, MA, USA.

Vinod Muthusamy (vmuthus@us.ibm.com) is a research scientist at IBM Research AI in Austin, TX, USA.

Aleksander Slominski (https://aslom.net) is research staff member in the Serverless Group in Cloud Platform, Cognitive Systems and Services Department at IBM T.J. Watson Research Center in Yorktown Heights, NY, USA.

© 2019 ACM 0001-0782/19/12 $15.00

Watch the authors discuss this work in the exclusive *Communications* video. https://cacm.acm.org/videos/the-rise-of-serverless-computing