A CTO’s Guide to Serverless Computing

Published 29 April 2022 - ID G00761697 - 12 min read

By Analyst(s): Arun Chandrasekaran, Craig Lowery

Initiatives: Digital Future; Digital Products and Services

Serverless computing redefines the way we build, consume and integrate cloud-native services. Enterprise architecture and technology innovation leaders, including CTOs, can harness its agility, elasticity and cost-effectiveness for emerging workloads.

Additional Perspectives

- Summary Translation: A CTO’s Guide to Serverless Computing (23 June 2022)

Overview

Key Findings

- There continues to be confusion on the definition of serverless computing and what products fall under it.
- Operational simplicity, platform elasticity and cost savings are the primary drivers for the adoption of serverless computing, while vendor lock-in and a low degree of operational control are the commonly cited limitations.
- Using serverless computing at scale requires organizational maturity and needs significant know-how and skills in the areas of software architecture, security and operations management.
- Beyond the public clouds, serverless architectures are now being used at the network’s edge to achieve just-in-time, personalized user experiences and for rapid processing of event data closer to the point of collection.

Recommendations

Technology innovation leaders who are CTOs, and who are investigating serverless computing, should:

- Choose "greenfield" use cases for serverless functions. Start with event-driven applications that are inherently "bursty" or highly parallelizable to maximize the benefits of the pricing model.
- Examine the impact of serverless adoption across different teams and instill a product engineering mindset and DevOps approach for serverless computing.
- Consume native services from your provider to fully benefit from a serverless architecture, but run a thorough proof of concept (POC) to validate your assumptions on application scalability, performance and cost of ownership.
- Follow cloud-native best practices: Use microservices architectures where appropriate, keep functions small and discrete, integrate security into the build and runtime, automate code deployment, and avoid "lift and shift" of code.

Strategic Planning Assumptions

60% of global enterprises will have deployed serverless function as a service (FaaS) by 2027, up from less than 25% today.

By 2027, 25% of serverless functions will run on the edge, up from less than 5% today.

Introduction

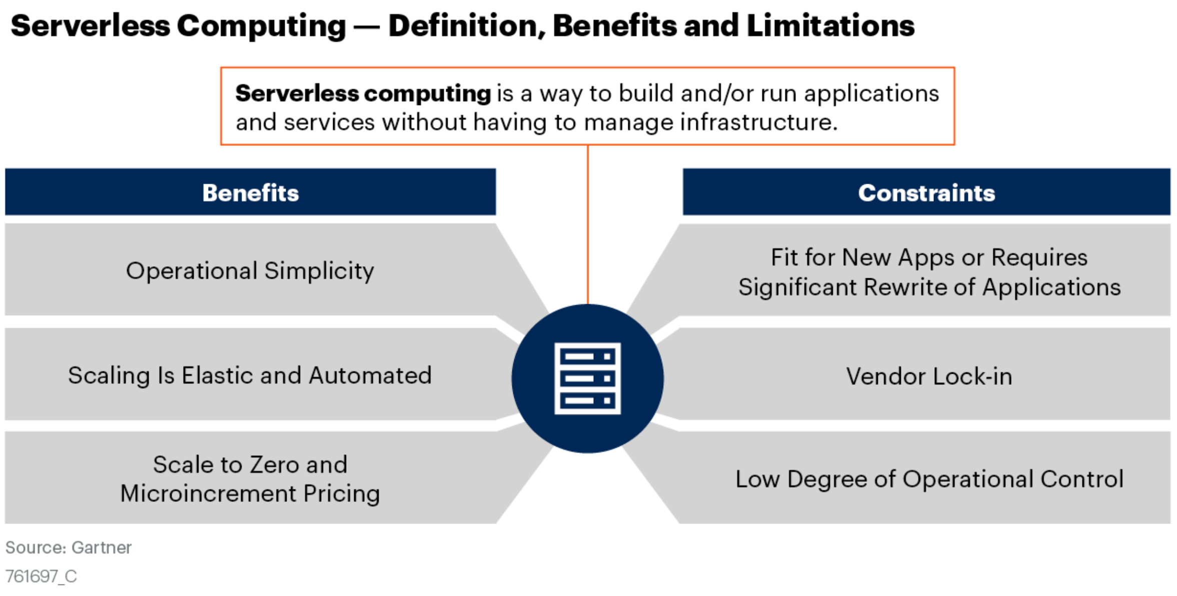
Serverless computing is a hot topic in the software architecture and cloud computing world. Despite its growing popularity, there is considerable lack of understanding in many IT organizations about what it is and what it can do. With this research, we define "serverless," dispel myths about it, outline the vendor landscape, offer best practices and demonstrate some use cases that are already generating business and technological value.

Analysis

What Is Serverless Computing?

Serverless computing is a way to build and/or run applications and services without having to manage infrastructure. The most prominent manifestation of serverless computing is serverless functions (fPaaS or FaaS). FaaS abstracts away the runtime environment, enabling developers to focus on application design and development. The following chart summarizes what it is, its benefits and limitations:

Figure 1: Serverless Computing — Definition, Benefits and Limitations


Serverless Computing — Definition, Benefits and Limitations
Serverless computing is a way to build and/or run applications and services without having to manage infrastructure.
Benefits
Operational Simplicity
Scaling Is Elastic and Automated
Scale to Zero and Microincrement Pricing
Constraints
Fit for New Apps or Requires Significant Rewrite of Applications
Vendor Lock-in
Low Degree of Operational Control
Source: Gartner
761697_C

The term "serverless computing" extends beyond FaaS. It encompasses any operational model where all provisioning, scaling, monitoring and configuration of the compute infrastructure are delegated to the platform. Examples of such services include AWS Fargate, Amazon Simple Queue Service (SQS), Amazon Athena, Azure Container Instances (ACI), Azure SQL database, Google App Engine, Google Cloud Run and IBM Cloud Code Engine. However, most discussions in this research on serverless computing will focus on FaaS, which is the most popular manifestation today.

What Are the Benefits and Limitations of Serverless FaaS?

The key benefits of serverless FaaS are:

- **Operational simplicity:** FaaS platforms obviate the need for infrastructure setup, configuration, provisioning and management. Hence, they have lower operational overheads than those in which developers build services directly on VMs or containers.
- **"Built-in" scalability:** In serverless FaaS, infrastructure scaling is automated and elastic, which makes it very appealing for unpredictable, spiky workloads. Hence, application scalability is most often limited by poor application design rather than any inherent limitation in the underlying infrastructure.

- **Cost-efficiency:** In most FaaS environments, you only pay for infrastructure resources when the application code is running, which exemplifies the “pay as you go” model of the cloud. Cost is a direct function of application design and code efficiency arising from best practices of the use of FaaS.
- **Developer productivity and business agility:** Serverless architectures allow developers to focus on what they should be doing — designing the application and writing code — and abstracts away most infrastructure aspects. It also enables business agility, where the time to market for new digital projects can be significantly shortened while simultaneously allowing for rapid experimentation.

These benefits must be balanced with the following limitations:

- **Architectural limitations:** Serverless architecture imposes some limits on the execution environment. Some are inherent, like cold starts (initialization latency), while others are artificial, such as the limits on function runtime and memory or cumbersome and inefficient use of shared code. The lack of server-side state support (data that persists in memory from function call to call), makes it less suitable for stateful and latency-sensitive workloads.
- **Application affinity:** With FaaS, application logic must be packaged as discrete functions, which are executed when triggered by events. This means existing applications must be significantly refactored to fit this packaging model, or new applications need to be written to fit these patterns. FaaS is better-suited to application design patterns such as microservices and event-driven architecture.
- **Skills gap:** FaaS requires a major shift in application architecture skills and best practices. These skills do not exist in abundance in the market, so teams typically pick up the knowledge as part of adopting FaaS. Although this may eventually lead to success, it may disillusion some early in the adoption phase and delay delivery of projects.
- **Vendor lock-in:** The leading FaaS implementations are proprietary to a specific provider, where organizations take advantage of native integration and tooling. While this is beneficial for shortening time to market and keeping things simple, if the application has to move from one platform to another, then it will have to be significantly reengineered.
- **Low degree of control:** The managed service model and runtime virtualization bestow huge benefits, but at the cost of little to no control of the service. The environment is a “black box” that must be used as-is.

What Are the Primary Use Cases for Serverless FaaS?

In general, serverless functions are suitable for projects that have functional components exhibiting at least a few of these characteristics:

- Tied to external events, such as receiving telemetry data
- Runs infrequently or is bursty
- Has scaling requirements that are highly variable or unknown
- Application can be packaged into discrete functions that are small and ephemeral, often lasting a few seconds
- Can operate in a stateless manner across invocations

Typical use cases that exhibit these characteristics are:

- **Cloud operations:** The most natural use case is to harness FaaS to execute operational code that manages the cloud environment.
- **Event processing:** Serverless functions are a natural fit for event processing. For example, a function could be triggered whenever an object is placed into a bucket in the object store such as looking for uploaded photos to then place into an image catalog, along with a thumbnail.
- **IoT platforms:** Serverless functions are a natural ingress point for IoT data. Each function captures the incoming data, then processes it in some manner, usually resulting in aggregating the data, storing it, or triggering a new event or generating a control signal directly back toward the edge.
- **Service integration:** A key value proposition of the cloud is to source the best service for a task, regardless of its origin, and knit it into a larger application context. Among the many problems associated with this model, ease of integration is one that FaaS solves neatly when used with RESTful APIs in combination with an API gateway.
- **Content delivery:** There are several emerging use cases in the edge such as API performance enhancement, content prefetching and rapid ad targeting.

What Serverless FaaS Solutions Exist Today?

Serverless functions are most widely deployed in public clouds today. Table 1 summarizes the key cloud-native serverless solutions available.

Table 1: Key Cloud-Native Serverless Solutions

Cloud Provider	Product Name	Languages Supported
Alibaba Cloud	Function Compute	C#, Go, Java, Node.js, PHP, Python, Ruby
AWS	Lambda	C#, Go, Java, Node.js, PowerShell, Python, Ruby
Google Cloud	GCP Cloud Functions	C#, F#, Go, Java, Node.js, Python, Ruby, Visual Basic
IBM	IBM Cloud Functions	Node.js, Java, Python, Swift, Go, PHP, .NET core and Ruby
Microsoft Azure	Azure Functions	C#, F#, Java, Node.js, PowerShell, Python, TypeScript
Oracle	Oracle Cloud Functions	Python, Go, Java, Node.js

Source: Gartner (February 2022)

There are open-source frameworks such as OpenFaaS and Fission for self-managed on-premises and hybrid deployments, although their maturity and turnkey value propositions tend to be questionable.

Serverless frameworks are becoming available in edge environments with a focus on minimizing the impact of cold starts and vendors leveraging their global content delivery network (CDN) to execute functions from edge locations that are closest to the end user. Cloudflare is notable as a pioneer, complementing its FaaS with datastores for stateful apps.

Table 2 summarizes the key vendors and products available at the edge.

Table 2: Simple Table

Vendor	Product
Akamai	EdgeWorkers
Amazon Web Services	Lambda@Edge
Azion	Edge Functions
Cloudflare	Workers
Cox Edge	Serverless Scripting
Fastly	Compute@Edge
StackPath	Serverless Scripting

Source: Gartner (February 2022)

How Do We Ensure That We Are Organizationally Well-Structured to Take Advantage of Serverless FaaS?

Serverless will impact several technology functions within the organization, with the most prominent ones being:

- **Application development:** Developers need to adopt a service-oriented development mindset because they also need to think about the security and operational aspects (such as high availability [HA] and observability) of their microservices application development. To reap the full advantage of serverless, put developers and operators closer together in a product engineering team so that they share responsibility for the development and operation of a software product throughout its entire life cycle.
- **Security and risk:** The biggest change that security and risk management leaders will have to adjust to is that they no longer own or control the OS, hypervisor, container and application runtime. Securing FaaS will force information security and risk professionals to focus on the areas that they can control. These include integrity and assurance of the code, access control, network connectivity, and minimizing the attack surface through real-time scanning and heuristic detection.

- **Infrastructure and operations (I&O):** FaaS doesn't entirely obviate the need for operations. Rather, it forces I&O teams to rethink IT operations from infrastructure management to application governance, with an emphasis on monitoring, cost management and ensuring that application SLAs are being met. Although functions can be deployed on-premises, such deployments are rare due to lack of rapid elasticity and absence of close integration with event sources such as data stores.
- **Enterprise architecture:** Enterprise and cloud architects have an opportunity to ensure the right compute options are chosen based on application architecture, workload longevity, elasticity, hybrid IT needs and degree of operation control needed, to name a few factors. Enterprise architects (EAs) need to work closely with business units to identify differentiated use cases for serverless, articulate its limitations and help in choosing an end-to-end tool pipeline that balances the agility of Serverless FaaS with adequate service assurance.

What Top Practices Exist for the Design and Operation of Serverless Apps?

The following are some key lessons learned from early adopters:

- Take advantage of FaaS's fast development characteristics early in the project life cycle to build a proof of concept (POC) to validate your assumptions about the application design, code, scalability, performance and total cost of ownership.
- Follow cloud-native best practices. Use microservices architectures where appropriate, keep functions small and limited to a single function, integrate security into the build and runtime, automate code deployment, and avoid "lift and shift" of code.
- Empower your product teams to be agile and iterative in development, and bring security and operations on as part of this DevOps product team to provide inputs early during the planning or application rebuilding phase.
- Do your homework on the security aspects of serverless deployments. Ensure that your existing cloud security posture management (CSPM) tool can provide risk visibility and configuration management for FaaS, as well as scan for vulnerabilities in real-time.
- Be sure to evaluate the costs of API gateway usage, which may be unacceptable for workloads with high rates of function invocation.

- Deploy tooling to gain visibility into functions, and rapidly troubleshoot functions to automate the continuous integration (CI)/continuous delivery (CD) pipeline. The tooling around local testing and application debugging is immature, but rapidly evolving.
- Create repeatable serverless patterns that can be reused across multiple applications and teams. This broadly distributes best practices and speeds adoption.

What Is the Future of Serverless Computing?

- **Greater versatility:** We expect providers to continuously improve on their FaaS offerings in terms of supporting more programming languages; offering better security; and providing monitoring, application debugging and local testing tools.
- **More serverless back-end services:** Expect more cloud platform services — databases, message queues and even container services — to be delivered in a serverless operational model.
- **Extension beyond the cloud:** Although FaaS and other back-end services have been primarily delivered as part of a public cloud platform, there will be increased adoption at the edge in the future.
- **Stateful workloads:** The inability to run stateful workloads (persistent state on the host) in FaaS is a key limiting factor for enterprise customers. Already, stateful FaaS is emerging with integrated key value stores and other forms of data stores.
- **Evolving open-source options and standards:** A key open-source effort for enabling self-managed FaaS platforms is Knative, which was originally created by Google. Knative is now being adopted by several other vendors in their efforts to develop serverless computing solutions on Kubernetes. Another effort to help enable industry-standard platforms for supporting data-intensive applications is the CloudEvents specification for describing event data. The goal of the project, which is hosted by the Cloud Native Computing Foundation (CNCF), is to enable event interoperability across services, platforms and systems from different vendors and service providers.

Evidence

This research is based on more than 100 Gartner client inquiries handled on this topic by the authors. In addition, detailed interviews were conducted with several early adopters of serverless computing on their deployment experiences. Many vendors profiled in this research have briefed the authors over the past two years.

Document Revision History

A CIO’s Guide to Serverless Computing - 28 April 2020

Recommended by the Authors

Some documents may not be available as part of your current Gartner subscription.

Top Strategic Technology Trends for 2022: Cloud-Native Platforms

Compute Evolution: VMs, Containers, Serverless — Which to Use When?

How to Navigate the Application Platforms Market Including Cloud-Native, Low-Code and SaaS

Innovation Insight for Cloud-Native Application Protection Platforms

---

Table 1: Key Cloud-Native Serverless Solutions

Cloud Provider	Product Name	Languages Supported
Alibaba Cloud	Function Compute	C#, Go, Java, Node.js, PHP, Python, Ruby
AWS	Lambda	C#, Go, Java, Node.js, PowerShell, Python, Ruby
Google Cloud	GCP Cloud Functions	C#, F#, Go, Java, Node.js, Python, Ruby, Visual Basic
IBM	IBM Cloud Functions	Node.js, Java, Python, Swift, Go, PHP, .NET core and Ruby
Microsoft Azure	Azure Functions	C#, F#, Java, Node.js, PowerShell, Python, TypeScript
Oracle	Oracle Cloud Functions	Python, Go, Java, Node.js

Source: Gartner (February 2022)

Table 2: Simple Table

Vendor	Product
Akamai	EdgeWorkers
Amazon Web Services	Lambda@Edge
Azion	Edge Functions
Cloudflare	Workers
Cox Edge	Serverless Scripting
Fastly	Compute@Edge
StackPath	Serverless Scripting

Source: Gartner (February 2022)