The Pros and Cons of a Serverless Architecture, and How to Prepare

In this e-guide

In this e-guide:

A serverless architecture allows developers to focus more on writing code, and spend less time on managing underlying resources. However, preparing for a serverless architecture will take considerable time and preparation, as you work to understand the benefits, caveats, and necessary tools.

Read this e-guide to learn more.

Next Article

In this e-guide

The pros and cons of serverless architecture

Chris Moyer, VP of Technology

The term *serverless* generates a lot of discussion. What exactly does it mean, and how can it help developers move from a monolithic architecture into a distributed one? Similarly, there is also confusion about the different benefits of containers and serverless architectures. Both architectures are modern approaches to application management, and each has specific benefits.

The best way to understand the difference between containers and serverless architecture is to look at the developer communities around each. Most documentation for Docker's container approach addresses issues that surround how to manage your infrastructure. The tools are designed to help more easily manage underlying hardware or virtual machines, and spread containers across multiple servers or instances in AWS. Documentation that addresses serverless frameworks and activity within the serverless community tends to focus on building serverless applications.

Fundamentally, serverless lets developers focus on writing code. There are still servers somewhere in the stack, but the developer doesn't need to worry about managing those underlying resources. While services like Amazon Elastic Compute Cloud (EC2) require you to provision resources for the OS and the application, a serverless architecture simply asks how many resources a single demand of your function requires. For example, a web testing suite might require 128 MB of RAM for any single website. Even if you deploy 10 million copies of that function, each individual one needs only 128 MB. They can even all run at the

In this e-guide

same time. Serverless focuses on what each individual request requires and then scales automatically.

Approaches to serverless development

There are several different approaches to serverless development. Most developers who transition from a traditional framework, such as Flask, Rails or Express, might choose to use a serverless framework, such as Chalice for Python or Serverless for Node.js. These frameworks are similar to the traditional ones, which help ease the transition for those developers.

Unfortunately, there are size and complexity limits to the single-framework approach, so developers who might build an old-style, monolithic application will quickly run into issues when they try to migrate that app to serverless. For example, a single AWS Lambda function can only be about 50 MB. This might seem large, but this also includes all third-party dependencies, as those must be included at deployment time.

Additionally, when a developer uses AWS CloudFormation, he will discover there is a limit to how complex the APIs can be. Therefore, he will need to split them apart once you have too many endpoints or operations. Furthermore, all of the same pitfalls of any monolithic service apply, so it becomes harder to upgrade, harder to maintain and a single-point-of-failure for the environment. However, with a single function, cold starts are

If executed correctly, a serverless architecture can save development teams time when pushing out new features and can scale nearly infinitely.

In this e-guide

easier to manage.

Microservices are a different approach to serverless development. In this case, you can still use a framework, but split the API into multiple microservices. This approach lets you share code between services via private packages that communicate via AWS Lambda invocations. Consider a scenario when a company operates an email marketing system which is comprised of several different microservices. Its application is hosted out of Amazon CloudFront, which uses one service to let users build a template, another service to let them pick the recipients and a third service that does the actual emailing. Each of those services is also split into separate microservices. The emailing service first builds a list of recipients in one function. Then, it passes the email template and recipient list along to another function, which splits that recipient list and passes each recipient, plus the email, to a third function to do the emailing.

Serverless functions are often chained together, which is a common pattern that can help mitigate the five-minute runtime limit, as well as the 50 MB size limit. In the email marketing system example, the first function, which handles building the recipient list, needs to have access to Amazon DynamoDB to pull down recipients. But it doesn't need to have the code installed to process the email template or send the actual email messages.

The last function, which does the actual emailing, doesn't need access to DynamoDB, but it does need to know how to build the template from the input data. But most importantly, none of these functions need to be exposed via Amazon API Gateway. Instead, that's handled through a separate service which simply takes a user request, authenticates it and then passes it directly along through an AWS Lambda call to the emailer stack.

For complex interconnected services, such as the aforementioned email example, developers can choose to use AWS Step Functions instead of connecting Lambda functions together by hand. This builds in additional support for error conditions, adds more retry logic and can

In this e-guide

automatically handle state and data transfer between functions. Each function is still completely isolated, but the state and transitions are all handled by AWS.

Tools for debugging serverless architectures

Traditionally, developers could simply log into the system, run the application, tail logs and test input to debug. In a serverless architecture, there is no server to log into, and running locally can be a lot more complicated. Some AWS plug-ins, such as Serverless Offline and SAM Local, offer support for running a majority of applications offline. However, these don't function so well when an authorization step happens in another repository or where there are multiple functions that need to be chained together. In many cases, developers must run their own stack for development and test and then push changes to a development AWS account.

There are several tools that can help developers and operations teams identify application problems and track down performance issues. AWS X-Ray can automatically trace issues with other calls to AWS. For most applications, it will only take a few lines of code to enable X-Ray. It can then show network graphs and point out issues with, for example, provisioned throughput on DynamoDB tables or Lambda concurrency limits. Console logs from both standard error and standard output within a Lambda application are directed to Amazon CloudWatch Logs, which can be ingested into an Amazon Elastic Search instance or interacted with directly through the API or AWS Management console.

There are also third-party tools and services that help with serverless tracking, such as IOpipe and New Relic. Also, there might be logs from other AWS services, such as API Gateway, that include valuable information when debugging issues. It's more complicated to actively monitor services because traditional monitoring tools, such as Pingdom, don't offer a way to test functions, just API endpoints. As a result, developers must build tools to run tests and expose those via APIs in order to use traditional infrastructure monitoring systems.

In this e-guide

The serverless tradeoff

Overall, serverless lets development teams focus more on the product and the output of an organization, but it does require more planning to handle testing and monitoring. Organizations that plan to use serverless should first build out a project map that helps them decide if they want to use a microservices architecture or rely on a single-function router to handle API requests. If executed correctly, a serverless architecture can save development teams time when they push out new features and can scale nearly infinitely. If developers skip advance planning and take precautions, it can lead to future problems.

Next Article

In this e-guide

Understanding the benefits of serverless functions

Zachary Flower, Freelance writer

The task of simultaneously managing server infrastructure and writing code is exhausting. For overworked developers, today's RESTful APIs and serverless platforms may be a match made in heaven.

Despite the name, serverless computing doesn't actually mean that there aren't any servers involved. It simply means that developers aren't required to think about the servers. This characteristic is what allows developers to build powerful, single-serve applications without having to deal with resource management. This certainly can make developers lives easier, while at the same time providing helpful abstractions away from the complexities of API integrations.

Enter the REST API

Serverless applications create a service-centric architecture out of the box, which means that the individual features of an application are both incredibly lightweight and highly targeted. While this mindset lends itself well to dealing with business logic, it is also incredibly powerful for integrating with external APIs.

Let's say, for example, you are building an application that needs to be able to charge users. However, your chosen API provider -- for example, Stripe -- offers much more functionality than you need. In keeping with your current serverless design, you decide to create a

In this e-guide

serverless function that does one thing and one thing only: charges a credit card using the Stripe API.

Rather than deal with the bloat of customers, subscriptions and invoices, the core application you create can request a single, targeted endpoint that tucks all those elements away for us. This effectively condenses third-party integrations into the functionality you need, rather than the functionality they offer.

While the above might seem like an over-abstraction, it demonstrates that third-party dependencies can be tucked into their own corners of an application, which allows developers to create serverless functions that solely provide the features you need.

That said, not every API requires abstraction. The flip side of the coin would be small, single-serve APIs that require the implementation of server-side technologies. While this usually isn't an issue with more monolithic applications, it can end up requiring more resources than leaner projects have available.

Take, for example, an application that requires a basic Slack integration. It may be the case that because of the way the application has been developed, creating a webhook for the Slack integration's "slash" command would mean spinning up a new server. Using a serverless architecture, developers can deploy ephemeral, dedicated serverless functions without adding any additional overhead to the core application.

Keeping it DRY

"Don't repeat yourself," or DRY, is a longheld mantra of the software development industry. While this is typically used to encourage developers to keep their code clean and maintainable, it applies to serverless application design as well. Let's take our two examples

In this e-guide

from above -- integrating with both the Stripe and Slack APIs -- and add a third API for good measure.

Assuming that our application relies heavily on these three APIs, there are three different ways that we need to build in order to maintain authentication and communication among each of them. While most APIs provide clean SDKs, that doesn't do much to ease the pain of working with more than one at a time. This creates complexities that can be difficult to manage within a codebase.

If each of those API integrations were abstracted out into serverless functions, then not only would the core codebase be easier to manage, but the way the codebase works with them would be cleaner as well. Rather than managing three separate methods of communication, we can reduce that down to just one function.

Beware of the caveats

It is important to note that the above scenarios focus on a very specific type of serverless infrastructure, called function as a service. This is the infrastructure made popular by Amazon Lambda. The other type of infrastructure is the one that completely removes all consideration of servers. These serverless platforms tend to offer highly vertical functionality, such as supporting only one language or only one type of application.

Additionally, sometimes a single function just isn't enough, and the metadata of the request is just as important as the request itself. This can lead to the creation of custom workarounds to keep things consistent. These workarounds often lend themselves to provider lock-in, which can cause issues down the line when you outgrow your current infrastructure.

In this e-guide

Next Article

In this e-guide

Why deploying APIs on serverless frameworks spurs innovation

Jan Stafford, Features Writer

Like Cinderella's slipper, serverless computing doesn't fit every foot in software development, but -- for the right application -- it offers royal opportunity.

Serverless computing provides a code and API deployment option for companies trying to reach more customers, grow their businesses and serve people in new and innovative ways -- without having to scale IT overhead. Serverless provides a deployment platform that, according to Gartner analyst Martin Reynolds, is "absolutely bulletproof" and guaranteed to deliver reliable, consistent results without losing information.

For software pro Chris Moyer, building and deploying APIs on a serverless framework has been a good step. Doing so makes the API lightweight on the client side, scalable on the cloud, easier to secure and less expensive.

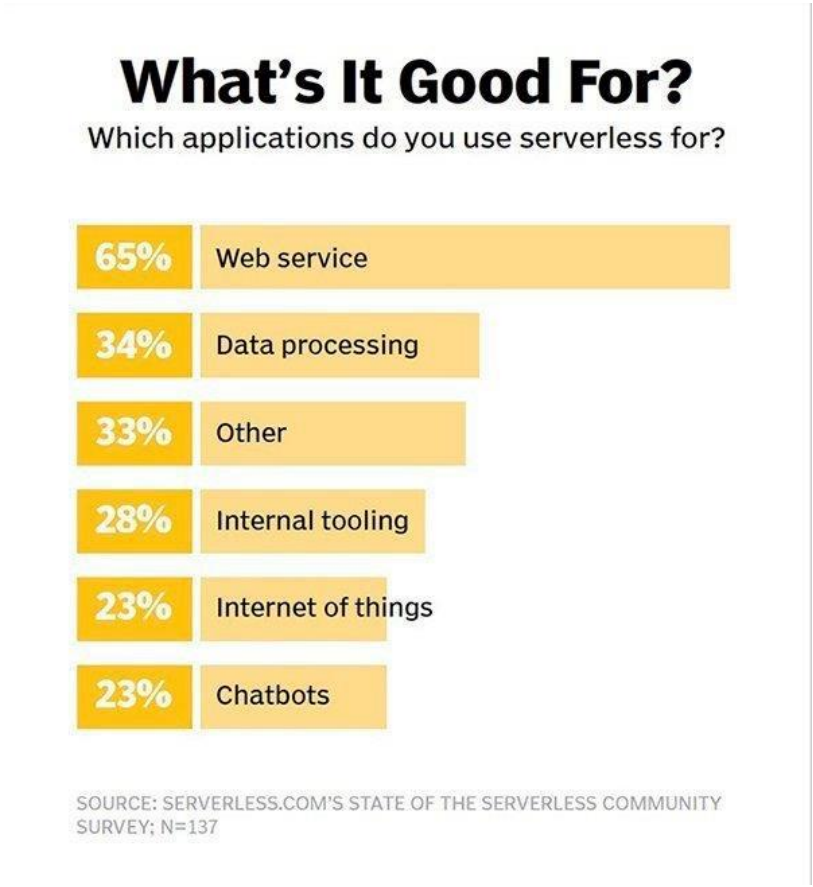
In his role as CTO of marijuana-license tracking and research firm Cannabiz Media, Moyer chose to deploy a serverless API that has to respond to requests quickly. That approach makes sense because the API doesn't need a cloud instance's distribution network to serve its front-end static pages.

Moyer chose to deploy another Cannabiz Media API in an Amazon Elastic Compute Cloud instance. "It's for running some back-end tasks that take longer to do, like two hours, as opposed to five minutes for the serverless API," said Moyer, who is also vice president of ACI Information Group, a social media and blog aggregation service.

In this e-guide

The moral of this story is, "Don't try to squeeze everything into serverless, but do take advantage of the benefits of serverless wherever it makes sense," Moyer said.

By deploying APIs on serverless frameworks, an organization can realize benefits to both its business and its DevOps process.



In this e-guide

'It just works'

Serverless is, first and foremost, a developer's tool designed to deploy code quickly, said Judith Hurwitz, CEO of Hurwitz & Associates, a research and consulting firm. "You're developing code with tools you know and deploying on a serverless framework without having to do the setup," she said. "Developers don't have to know how it works. It just works."

You're developing code with tools you know and deploying on a serverless framework without having to do the setup.

CEO, Hurwitz & Associates

Judith Hurwitz

Serverless APIs don't have to fit into an enterprise's existing architecture, said Gartner's Reynolds. Indeed, deploying a serverless API doesn't require building a serverless architecture, either. Instead, most developers use lightweight packaged tool sets called *serverless frameworks* to deploy APIs.

"Serverless APIs are neutral on the architecture and framework," Reynolds said. "They are just little functions that you call. They are part of your application architecture, and they run in the serverless framework." Size and function are the key criteria for choosing serverless APIs. "If an API is small and just built to do a call to a back-end system, like saying what to do at a certain time, deploying on serverless makes sense," Hurwitz said.

Fixate IO analyst and developer Zachary Flower avoids deploying long-running or complicated processes on serverless frameworks. "Working with APIs that can be easily

In this e-guide

abstracted into simple function-style endpoints will go a long way towards maintaining sanity," Flower said.

Which APIs fit serverless?

Single-feature APIs with small footprints work well for specific serverless functions, Flower said. Monolithic APIs are out. Treat each endpoint as a simple action, rather than a sentence or paragraph; in other words: *Do x* as opposed to *do everything*, Flower explained. An example is the Twilio REST API that simply requests an SMS. "These are simple abstractions that demonstrate how to best consume an API in serverless," Flower said.

Any API function that can return in less than 30 seconds works in serverless computing, Moyer said. There's a 30-second timeout in the typical API gateway. "Basically, anything that takes longer than 30 seconds is questionable," Moyer said.

There are other functions that may be squeezed into the serverless computing mold, but let the user beware.

Don't try to put serverless on top of your existing virtual machine infrastructure, Reynolds advised. Serverless is all about building as little overhead as necessary. "Go all in, and build it in a serverless framework," he said.

Four timely enterprise uses for serverless computing

While APIs are an excellent entry into serverless computing, plenty of other viable uses exist. Contributors and experts suggested that decision-makers explore these four uses for serverless frameworks and function as a service (FaaS):

In this e-guide

- **Microservices.** Tom Nolle, president of consulting firm CIMI Corp., sees microservices deployment as the best use case for serverless today. "Properly designed microservices are little functional atoms that can be made stateless easily, scaled easily and used to compose apps easily," Nolle said.
- **Event processing**. In Nolle's opinion, this is the broader mission of serverless and FaaS. In particular, the ever-expanding adoption of IoT-related event-driven applications will drive the need for serverless computing's flexibility and ease of deployment. Deploying an event-driven application on IaaS would waste compute resources, as the instance would only be used when an event occurs.
- **Retail**. Retailers such as Nordstrom are early adopters of serverless computing. "It's a means of opening the retail product line and distribution channel to new customers," Gartner analyst Martin Reynolds said. He expects Amazon to use FaaS to build out the online business of Whole Foods, which it acquired in 2017. "With serverless, they will be able to scale the Whole Foods business without increasing the spend on IT," he said.
- **NoOps applications**. Developers can deploy apps on a serverless framework without any involvement from operations, Reynolds said. Serverless offers an automated package of the ops infrastructure and tools used in the software application lifecycle management process. Developers can even test serverless code in the cloud without going through the operations workflow. Microservices could work well here, as would any software functions that produce semi-static information specific to users, Reynolds said.

The list of plausible enterprise uses for serverless is much longer, of course. Today, organizations are deploying static and semi-static information and webpages, and single-page and hosted static websites, on serverless frameworks.

Powerful, real-time socket APIs are better suited to traditional infrastructure, as they will quickly erode the cost savings of serverless, Flower said. For example, in API design, using a function as a service (FaaS) approach rather than a serverless framework for overly flexible API infrastructures like GraphQL can become more trouble than it is worth, he said.

In this e-guide

Moyer had a right-sizing problem with serverless APIs when he found that his content delivery network (CDN) limited resources per cloud formation template. He had set up 15 different APIs via a serverless framework. "I wanted to add something that would allow people to fetch other users in their accounts," he said. In time, those APIs hit the CDN's 200-endpoint limit. "I could no longer add a new API endpoint. I had to split the APIs apart."

Benefits of serverless APIs

Building and deploying the right type of API on serverless platforms brings many benefits, including reducing repetitive, low-value tasks in software development. Even better, it relieves worries about scaling and managing APIs. For a business, serverless computing can lower development costs and increase monetization opportunities, but without the risk of vendor lock-in.

Some key benefits to deploying serverless APIs include the following:

Security. Serverless APIs run in a completely trusted and secure environment, Reynolds said. Serverless is a way to deploy proprietary APIs and code in almost any environment without anybody being able to see what's going on. It promises, and has so far delivered, that "nobody can actually attack that code," he said.

Scalability. The traditional way to build safe applications doesn't scale well, Reynolds said. Serverless scales automatically, which pays off when APIs and apps must scale to meet the needs of many customers at once. Reynolds cited Nordstrom's use of Amazon Web Services' Lambda serverless platform to scale its customer requests for product recommendation and reduce response time from minutes to seconds. "You can't keep adding mainframes to do this stuff," Reynolds said.

In this e-guide

Cost. Users only pay for the compute time they use. For example, Cannabiz Media has clients nationwide and a very random traffic pattern. There are spikes first thing in the morning in each U.S. time zone, but traffic can also spike in the early evening. "It doesn't make sense for us to run even one server overnight when most of the time there's not going to be anyone using it," Moyer said. With the requests routed through a serverless API, the company is paying per request, not for the time people aren't using it.

Management. It's automated. "You don't have to manage anything," Reynolds said. "If there need to be another 10 instances in Canada, boom, they're there," Reynolds said. "You didn't even have to know about it. They're just there as opposed to all the paperwork you'd have to do if you needed a new server."

Monetization. Businesses can sell their APIs as a serverless function, rather than taking the longer route of patenting it, Reynolds said. For example, a developer creates a black box function that helps a retailer sell more frozen peas. The developers could provide it as a serverless interface available in a cloud provider's API library and then charge for a call of that function, Reynolds said. "Serverless opens the opportunity to monetize algorithms, APIs and other ideas that you couldn't see how to do before," he said.

No lock-in. While Amazon delivered the first modernized cloud-based serverless offering, it's not the only game in town, Moyer said. Today, changing serverless framework providers is simple. "A DevOps team can choose to re-architect for serverless using a cloud provider's FaaS platform or taking a simple approach by deploying software on a serverless framework," Moyer said.

Serverless frameworks, FaaS or stateless? What's the diff?

The definitions of types of serverless computing are moving targets. FaaS, serverless architecture and serverless frameworks are frequently referred to as one and the same. While

In this e-guide

each refers to an event-driven, pay-per-use cloud service, knowing the differences among them can be important.

Serverless or stateless? In some IT circles, serverless is synonymous with stateless. In both approaches, software development takes place entirely in the cloud, where automated infrastructure and tools needed to run application code or services are provided. Classic serverless refers to functional computing or stateless microservices, said Tom Nolle, president of consulting firm CIMI Corp. In these application architectures, developers create a specific model of transient code that can be scaled or replaced at will.

Serverless architecture. Keeping it simple, Gartner analyst Martin Reynolds sees a serverless architecture as a model for how a development team builds and deploys applications. Most application functions take place on the front end. There's no need for running an always-on server for these on-again, off-again activities.

Serverless framework. A serverless framework is a product for deployment of serverless applications. In essence, the framework is a command-line interface tool. An automated tool set facilitates building and deploying web, mobile and IoT applications on event-driven compute services.

Function as a service. FaaS refers to specific cloud service offerings that provide capabilities similar to serverless frameworks, but typically have closer integration with the cloud compute provider. Amazon's Lambda product is a widely known FaaS. Because of these roots, FaaS and serverless are considered synonyms by many, but FaaS and serverless frameworks are not. With FaaS, developers may be required to create some of the server-side logic, but most offerings include a serverless architecture. Some do not. Docker, for one, provides FaaS but requires the user to manage the underlying infrastructure.

In this e-guide

This is the year to experiment with serverless, and APIs are good test cases. Reynolds suggested doing some trial runs to find internal APIs and applications that can deliver better value with serverless than with fixed IT assets. "A business using serverless computing can dramatically increase its ability to scale and significantly reduce the cost of dealing with variable workloads," he said.

Next Article

In this e-guide

Three ways to prepare applications for serverless platforms

George Lawton, Contributor

Major cloud vendors, including AWS, Microsoft and Google, have all rolled out serverless platforms that enable users to weave together simple, or even highly complex, applications on top of a collection of functions. But to run cloud applications successfully on one of these platforms, development teams need to evolve their design processes.

A few key principles that developers should follow to build serverless apps include:

1. Develop small and discrete application components.
2. Implement stateless functions.
3. Plan for ephemeral functions with a short lifespan.

1. Smaller, discrete components

Serverless functions, by nature, are intended to perform a single task. This requires teams to keep their functions small and to orchestrate their development processes around these separate application components.

A good practice is to break up functions, based on the types of events that trigger them, into their own repositories. Basically, developers should design each function from the get-go to respond to one event. If multiple functions are triggered by the same event, then include them in the same repository. This makes it easier for developers to identify functions based on the events that launch them.

In this e-guide

Each separate function should also include configuration data to ensure consistency when it launches in development, testing and production environments. It's also important to declare and isolate dependencies within each function, using tools such as npm and webpack.

2. Go stateless

Development teams should consider implementing functions as stateless services, in which the state of a function is stored outside the application itself. You can configure functions to reference data stores and databases that are managed as part of the configuration data and stored with the function in the repository.

Some serverless platforms offer local storage that persists between function calls. While developers might be tempted to use this for more complex functions, this local storage doesn't always hold between function calls and, even if it does, might not work consistently.

Try to use a backing service to store stateful data in a database, separate data store or cache. A backing service refers to any external service a function consumes over a network as part of its normal operation, including caching services, like Memcached, data stores like CouchDB or databases like MySQL.

3. Plan for ephemerality

One of the benefits of serverless functions is that they can start up in as little as a few milliseconds -- compared to several seconds or minutes for larger apps that run on PaaS, containers and VMs -- and stop just as quickly. This makes it easy to scale up a large number of functions in response to demand or spread computation across functions that run in parallel and then spin them back down once the process is complete.

In essence, serverless apps are more ephemeral in nature.

In this e-guide

This means, to fully realize the benefits of serverless platforms, developers need to remove many of the components that they commonly bundle with applications from the core logic of their functions. Developers should:

- prewarm the data sources required by functions;
- set up external logging and tracing infrastructure; and
- manage security.

Prewarm

To prewarm a function, you set up the infrastructure required for the function to communicate with databases in advance of deployment.

To do this, separate the function handler from the core logic. Much like idling a car to improve how it runs on a cold day, idling the event handler ensures that, as soon as it is invoked, the function can operate at peak performance. Tools such as AWS CloudFormation and Azure Event Hubs support the prewarming process.

External logging and tracing

Functions need to be able to shut down quickly and gracefully. Although serverless platforms include some built-in native storage, it's important to configure an external logging and tracing service, built into the serverless framework itself, or to set up custom tools to gather and store logging data from functions as they run.

Tools such as AWS X-Ray and Azure Application Insights can help with logging and tracing.

Security

Serverless apps present another potential attack vector for hackers. Without proper security measures in place, hackers might be able to spin up functions and access sensitive data.

In this e-guide

Developers should configure security settings for classes of functions to allow for minimal access to other apps required to execute a particular task.

Next Article

The pros and cons of serverless architecture

Understanding the benefits of serverless functions

Why deploying APIs on serverless frameworks spurs innovation

Three ways to prepare applications for serverless platforms