Serverless Architectures: Everything You Need to Know

by Mike Chan | Mar 13, 2017 | Cloud Computing | 0 comments

What is serverless, what serverless frameworks are available, who uses it, and what are its pros and cons?

Serverless is an increasingly popular approach to software development that is set to explode in 2017.

The serverless movement started with the launch of AWS Lambda in 2014, and many other cloud providers followed by launching their own serverless offerings.

With any "hot" technology comes questions, such as:

- What exactly is serverless?
- What cloud providers provide serverless services?
- What companies use this approach?
- What are the pros and cons of serverless?
- What's the future of serverless?

Let's jump right in and answer these questions.

Definition of Serverless

Thoughtworks defines serverless as “an approach that replaces long-running virtual machines with ephemeral compute power that comes into existence on request and disappears immediately after use.”

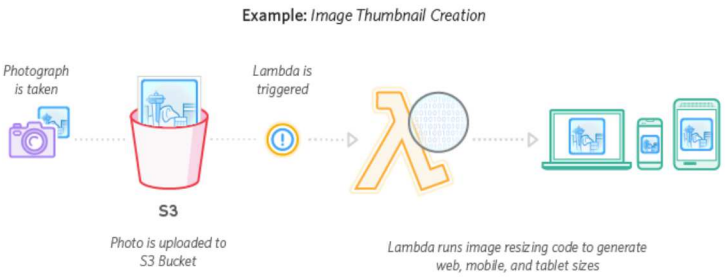
However it’s defined, the term “serverless” is actually a misnomer.

There certainly are servers involved in running your app; it’s just that you’re not managing these servers (your cloud provider takes care of that), and they’re not always running.

Rather, your application alerts the cloud server when it should execute an action, and when that action is complete, the server essentially stops running until another action is requested.

For instance, let’s say you build content management software and one of your users uploads an image to be included in a blog post.

If you used a serverless architecture built on AWS Lambda, when that image gets uploaded to an S3 bucket, an event can be triggered to resize the file so it looks good across all devices, and another event can fire to compress the image to save storage space. The servers that power these actions would not be active until the events are triggered by your users.



AWS Lambda File Processing – Image Courtesy of AWS

This approach is also called Function-as-a-Service (FaaS) and it allows developers to focus on building their applications instead of messing around with server infrastructure. And because the servers only run when necessary, serverless apps cost less to run.

It's an option that has recently seen a lot of momentum and might be a good option for the next app or website that you build.

Who Offers Serverless Frameworks?

There are a growing number of serverless frameworks available for you to choose from.

AWS Lambda, launched in 2014, was one of the first serverless frameworks available and is the most mature and developed. Lambda supports Node.js, Java, Python, and C# (.Net core) and seamlessly integrates with many other AWS offerings.

Most of the other major cloud computing providers have their own serverless offerings.

Google Cloud Functions is available in an alpha release, Microsoft Azure Functions launched in 2016, and OpenWhisk is an open source serverless platform run by IBM.

Other serverless options include Iron.io, which is an open source platform that can run on public, private, and hybrid clouds; Webtask, which is built by Auth0 and integrates its authentication platform; and the eponymous Serverless, which isn't a stand-alone framework but is a platform that runs on AWS Lambda and makes it easier to deploy and manage the multiple services that a serverless application may need, all in one codebase. (**If you're interested, check out our introductory tutorial on how to use Lambda with the Serverless framework.**)

Serverless Case Study – Netflix

Some of the largest companies in the world leverage serverless to power their products.

Netflix is one of the pioneers of using serverless architectures to deliver almost 7 billion hours of video to over 60 million customers around the world.

They implemented AWS Lambda to:

- Encode media files – they set up rules that trigger events when media assets are uploaded and moved through the encoding process. Events are fired to track, aggregate, validate, tag, and publish these media assets to their content delivery network.
- Back up files for disaster recovery – Lambda rules decide what files need to be backed up and copied to offsite storage, validate that these backups have been executed, and raise flags in the case where files have not been properly backed up.
- Secure their assets – Lambda helps them validate that their infrastructure is secure and fires events when unauthorized instances appear.
- Monitor their environment – Lambda events trigger the creation of new alerts and dashboards so Netflix can easily and auto-magically monitor the ever-changing state of their infrastructure.

Netflix has been extremely innovative in its use of serverless architectures to power their products at scale.

Pros and Cons of Serverless

Serverless provides many benefits to your development team, as well as some cons that you'll need to consider before implementing it.

Let's start with the positive.

Pros of Serverless

You only pay for what you use

The first pro is that you don't have to pay for idle server time, and only pay for the time that your app uses the server.

As mentioned prior, the server only runs when your application tells it to execute certain tasks, and that's the only time you need to pay for it. When those tasks don't need to be run, the server stops running until another task needs to be completed.

This is a very cost-efficient way to run an application and can save you a lot of money in the long term.

Your app is elastic

With a serverless architecture, your app can automatically scale up to accommodate spikes in traffic, and scale down when there are fewer concurrent users.

Let's say that you've built a ticketing platform for concerts and sports events.

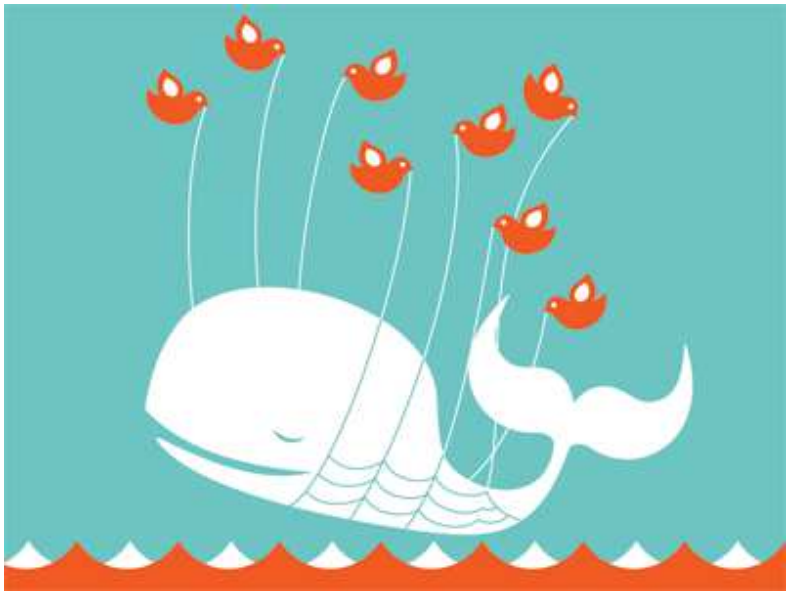
Traffic might increase significantly each time you release tickets for popular bands'

concerts or sports playoff games. But then the volume might normalize at the times when new concerts or events aren't announced.

A serverless architecture allows your application to quickly scale up to handle the demand of rabid fans searching for tickets, and scale down during the other times of lower traffic.

This will help cut down costs, since you'll only pay for what you use.

Maybe more importantly, the performance of your platform will increase and your customers won't ever have to deal with a slow, unresponsive website or see the fail whale.



Fail Whale – Image courtesy of Xiaobin Liu on Flickr

Less time and money spent on managing servers

Serverless architectures will allow you to spend less money and time managing servers, because most of the work will be done for you by your cloud computing provider.

The elasticity of a serverless architecture will alleviate the need for your IT team to manage the expansion and contraction of server capacity, because the serverless architecture scales automatically to meet the demand. No more guessing and much less worrying.

And in the case that something goes wrong and a server goes down (which is extremely rare), it's on your cloud computing provider, not your IT team, to figure out and fix the problem. While having your IT fate in the hands of others might seem like a bad thing, cloud providers are the very best at dealing with these kinds of issues.

Thus, you can focus on what's important – the design and development of your app and running your business.

Reduces development time and time to market

Serverless architectures can have a significant impact on the time it takes for you to get your product to the market.

Your developers don't need to worry about deploying, managing, and scaling servers. There are no operating systems they need to select, secure, or patch.

They can simply concentrate on building the best product possible, as fast as possible.

This will help get your products to market much more quickly so you can serve your customers better.

Fits well with microservices

Microservices is a popular approach to development where engineers build modular software that's more flexible, scalable, and easier to manage than its monolithic counterparts.

And serverless architecture fits very well with microservices.

With the microservices approach, developers can work autonomously and in parallel to build smaller, more loosely coupled pieces of the whole software pie. Because these smaller pieces are less dependent on each other, they can be changed faster and scaled independently.

One issue with microservices is that each developer may need to spin up their own instance of infrastructure to build their portion of the product. Each developer will need to worry about provisioning servers, scaling for high traffic, managing costs, and more. And across all developers involved, this can result in a lot of overhead, high costs, and a ton of wasted time.

Serverless addresses many of these concerns.

Developers don't have to worry about provisioning and managing servers, each microservice can scale almost instantly, and the team can build each piece of the product faster.

Check out this Quora thread for more insight into how serverless and microservices work together.

Want to chat about how you can leverage a serverless architecture in your next app? Just click here to contact us.

Cons of Serverless

With any new technology comes a few drawbacks that need to be considered and addressed. Here are some cons of serverless that you should think about.

No real roadmap or standards

Serverless is still in its infancy, and like any early-stage technology, there aren't any true standards or roadmaps to follow.

Noted serverless expert and Serverless Conf co-organizer Anthony Stanley quoted, "The most striking thing for me is how everyone seems to be doing it differently. There doesn't seem to be one blueprint that everyone is adopting."

Each of the cloud companies who provide serverless architectures may take a different approach. There may be varying methods of integrating different programming languages and frameworks. And different types of applications will use serverless in different ways.

Though serverless is progressing quickly, it's still the Wild West out there, so be prepared to adapt to many changes.

> **"The most striking thing for me is how everyone seems to be doing it differently. There doesn't seem to be one blueprint that everyone is adopting." – Anthony Stanley**

Higher latency in responding to application events

The serverless architecture isn't a good fit for all types of applications.

Because the server sits idle until a certain application event occurs, it may take some time to "wake up" and execute the command.

For instance, if you use AWS and your function hasn't been run in a while or you've made changes to your code, Lambda may have to create another container, which is definitely slower when compared to using an always-on virtual machine.

There are a couple of repercussions due to this.

First of all, serverless architectures may not be a good fit for applications where page load speed is absolutely essential, such as e-commerce, social media, and search sites.

Also, you may have to design your architecture so that your function remains in an active state by sending periodic requests to it (a "keep-alive ping", as one of our engineers calls it). This will help avoid the "cold start" problem and make your application run faster.

Not all programming languages are fully supported

Existing serverless frameworks support many programming languages, but there are a bunch of popular languages where implementing serverless is much more difficult.

AWS Lambda directly supports Java, C#, Python, and Node.js (JavaScript), Google Cloud Functions operates with Node.js, Microsoft Azure Functions works with JavaScript, C#, F#, Python, and PHP, and IBM OpenWhisk supports JavaScript, Python, Java, and Swift.

But there are many other popular programming languages, like C++, Golang, Scala, Clojure, and Haskell, where you'll have to work much harder and jump through more hoops to set up a serverless architecture.

Tools like Iron.io are trying to solve this problem, but many languages still lack full support from the major serverless players.

Few engineers are truly serverless experts, so hiring is expensive

Another drawback of being in such an early stage is that there is a lack of true serverless experts. Thus, hiring these developers can come at a steep price.

Of course, architects and developers working with traditional cloud technologies can certainly pick up the skills needed to build serverless architectures. But like anything else, there is a learning curve, and the technology continues to change very rapidly, making it even harder to keep up.

As serverless' popularity rises, so will the number of experts. Right now they are few and far between, so they can be tough to find and expensive to hire.

Vendor lock-in

Vendor lock-in is one of the primary concerns of moving to the cloud, and serverless is no different.

The back-end of your serverless architecture is completely managed by your cloud computing provider, and if you decide to move to another cloud platform, you'll likely have to make major changes to your application.

Not only will you have to port your code, you'll also have to consider changes to other parts

of your application such as databases, identity and access management, storage, and more.

That can take a lot of additional time, money, and resources.

Inefficient for long-running tasks

Because of its event-based nature, a serverless architecture might not be the best fit for applications that execute long-running tasks.

If you're building an online game where gamers can play for hours on end, the event-based nature of serverless wouldn't be a good fit.

If you're performing a big data analysis and need to query a large data set, you may be better off using traditional virtual servers.

If your application calls APIs that may take a while to return, a serverless architecture might inhibit those calls from being completed.

Most serverless providers have a time limit on how long a function can run. For instance, AWS Lambda and Microsoft Azure Functions don't allow you to run a task for longer than 5 minutes; after that, the task will be terminated.

This will either force you to either create multiple coordinated functions or opt for using a virtual server to properly execute your long-running task.

Additionally, running long tasks with a serverless architecture might wind up being more expensive than using a virtual machine or a dedicated server.

There are many pros and cons that you'll need to think about and understand before you incorporate serverless architectures into your app. Make sure that serverless fits well with your particular application and situation before implementing.

The Future of Serverless

Serverless will play an integral role in the future of software development.

An increasing amount of companies will continue to leverage serverless architectures to bring products to market faster and cheaper, and gain a competitive advantage over those who don't.

More standards will be created and more programming languages will be supported, so more developers can easily use the platform and become proficient with the technology.

Problems with latency and efficiency will continue to be addressed.

In short, the future is extremely bright for serverless and this approach will be around for a long time. Or at least until the next hot architecture comes by. 🙂

What are your thoughts about serverless architectures? Are you thinking about going serverless for your next application? We'd love to hear your thoughts in the comments.

Like this post? Please share it using the share buttons to the left! Then join our mailing list below, follow us on Twitter @thorntech, and join our Facebook page – for future updates.

Serverless Architectures: Everything You Need to Know

AWS Tutorial: Intro to using Lambda with the Serverless framework

Blog categories

AWS

Case Study

- Enterprise Software Development
- Mobile App Development

Cloud Computing

Enterprise Java

Enterprise Mobile

Featured

Jobs

Mobile

Mobile Analytics

Mobile Apps

- Ultimate Mobile App for...

Mobile Commerce

Mobile Marketing

Mobile Web

News

Portfolio

Tutorials

Web Development

Week in Tech

Thorn Technologies LLC
9175 Guilford Rd
Suite 212

Columbia, MD 21046
info@thorntech.com
(410) 429-0255

- About
- Services
  - Cloud Computing
  - Onsite AWS Training
  - Enterprise Software
  - Mobile Apps
  - Websites
- Success Stories
- Blog
- News
- Careers
- Contact Us