STACKERY

Managing the Serverless Application Lifecycle Safely and Intuitively with Architecture Design and Deployment Tools

EXECUTIVE SUMMARY

Serverless technology offers the possibility to dramatically reduce IT costs while allowing engineering teams to develop and deploy applications significantly faster than with either in-house servers or traditional cloud-based servers. However, the out-of-the-box set-up from serverless providers makes architecture design and implementation tedious and error prone while monitoring and debugging is close to impossible, making it difficult to justify using serverless technology for customer-facing applications. With the right tools, however, engineering teams can take advantage of serverless's potential without compromising the ability to deploy safely, monitor application performance and recover from a crisis.

WHAT IS SERVERLESS AND WHY COMPANIES SHOULD SWITCH

Serverless technology refers to a type of cloud computing in which the business or individual user does not provision servers directly, instead depending on a cloud service provider to provision and manage computing resources on an as-needed basis. Serverless technology–which absolutely relies on servers, even if the end user doesn't need to worry about them–allows companies to pay for only the computing time that their application actually uses.

In a serverless system, applications that are not in use are dormant. When an application is called, the functions run, serve the application and then go dormant again once the application is finished.

WHY MAKE THE SWITCH?

Lower IT costs

With serverless technology, you only pay for the computing resources you actually use. Companies running either an onsite data center or using traditional cloud computing need to provision enough server capacity to handle peak usage. Inevitably, though, most of the time the company would be paying for more server capacity than is actually needed. Serverless eliminates wasted computing resources and, in most use cases, dramatically reduces IT costs.

Faster application development

Serverless technology lends itself to quick time-to-market and applications developed by small teams. It's not unreasonable for a team to develop and deploy a serverless application within days of determining a business need. "Serverless is a game-changer

in how applications can be developed quickly," says Patrick Steger, owner of IDMA Solutions, a custom application development company. The agility in bringing ideas to the market has the potential to improve how well your business can respond to customer needs and to general market conditions.

Less engineering overhead

In a traditional computing world, engineers need to spend a significant amount of time provisioning and configuring servers. The time engineers spend managing servers is time that they are not working on projects that deliver value to customers. Serverless technology allows engineers to focus more of their time on creating value and less on managing the backend.

EPHEMERAL FUNCTIONS IN SERVERLESS

One of the key challenges in testing, debugging and monitoring serverless applications is the impossibility of logging in to a server somewhere and poking around to see what went wrong. Serverless functions run and then disappear. "All you have is the data that you collected while it was running," says Sam Goldstein, director of engineering at Stackery. If logs aren't collected in real time, as the application is active, information about everything from cold start times and other performance information to errors, error traces and failures is lost permanently. It's entirely possible to work directly with the tools provided by Amazon Web Services to correctly set up all of the necessary logs—at least in theory. In practice, it takes hundreds or thousands of clicks, making correct set-up unlikely.

The ephemeral nature of serverless makes logs even more crucial than in a traditional computing environment. It simply isn't possible to recover information after the fact if the configuration wasn't correct at run-time. Tooling is the only realistic way to ensure that performance and error data is collected across the board.

CHALLENGES FOR ENTERPRISE ADOPTION OF SERVERLESS

Serverless technology is catching on relatively rapidly, considering that Amazon Web Services launched Lambda in 2014. As a new technology, however, there are a number of challenges related to successfully deploying and maintaining serverless applications, especially business-crucial applications and any public-facing applications.

Many of the challenges in developing, testing and monitoring serverless applications stem from the fact that serverless applications are ephemeral. Traditional applications are essentially static. If there is an error, engineers often diagnose

the problem by logging in to the server after the fact and looking around to see what went wrong. Serverless applications, however, run and then disappear—and there's no server to log into, anyway. If you don't have error logging properly configured, it can be impossible to diagnose or recreate the error.

In addition, the tooling available for serverless applications is still relatively immature, and the solutions offered by serverless service providers like Amazon Web Services Lambda are difficult to use. Engineers could find themselves freed from server management but spending at least as much time configuring the serverless environment, managing deployments and manually searching through logs to debug their applications.

The following are some of the challenges to adopting serverless technologies in each phase of the application lifecycle:

Development and Deployment

After defining the application's business logic and conceptualizing the application architecture, the next step in building a serverless application with AWS Lambda is to define your architecture in CloudFormation. This is a laborious process that involves manually defining every aspect of your infrastructure using JSON. "In Amazon Web Services, you need to know how to provision a dozen different fundamental resources, like gateways, route tables, private and public subnets that are spread across multiple availability zones," explains Chase Douglas, co-founder of Stackery.

It's a slow, error-prone process that's also unfamiliar to many engineers who are new to serverless. The combination of hundreds of human steps, a steep learning curve and un-intuitive design make mistakes almost inevitable. The sheer amount of time it takes to correctly set up the application makes it impractical to set up a testing environment, while the opportunity for errors means that even if a test environment were set up, it would likely have slight configuration differences due to human error.

> "In AWS you need to know how to provision a dozen different fundamental resources that are spread across multiple availability zones" - Chase Douglas

Shipping the code to the production environment comes with its own difficulties. Unlike in a traditional server environment, development teams—not operations teams—are responsible for code deployment. This shift in responsibilities requires both a mentality shift and another learning curve—again, increasing the risk of error. This is further complicated by the fact that

serverless applications are generally complex. Any given application could have hundreds of functions that each need to be packaged correctly and shipped to the production environment. This deployment process could take an engineer days. In addition to being slow, the manual nature of deployment using Amazon Web Services' raw tools increases the risk of errors and makes repeatability elusive.

Testing

Testing a serverless application is fundamentally different from testing a traditional application run on either locally managed servers or in a more traditional cloud computing environment. In those cases, it's possible to create an environment on one laptop that is similar enough to the production environment to run tests on. That's not possible when developing a serverless application.

"You don't have a full replica of Amazon Web Services on your laptop," says Sam Goldstein, director of engineering at Stackery. Serverless applications generally have complex dependencies and are based on a large number of interrelated microservices. "You tend to be relying on cloud provider instructions in terms of how you pass messages between microservices or coordinate work between a cluster of things and there are certain advantages to that, but one of the disadvantages is it becomes very challenging to reproduce that or test that locally."

> "You tend to be relying on cloud provider instructions in terms of how you pass messages between microservices or coordinate work between a cluster of things and there are certain advantages to that, but one of the disadvantages is it becomes very challenging to reproduce that or test that locally." — Sam Goldstein

The only viable way to test serverless applications is to have a separate testing environment hosted with the serverless service provider.

This strategy depends on the ability to set up the test environment identically to the production environment and being able to deploy both the initial code and any changes in an easily repeatable way. Without additional tooling, both environment management and code deployment in AWS Lambda is too error-prone for reliable testing in a test instance.

Monitoring

In a serverless environment, monitoring is another responsibility that is shifted from operations teams to development teams. Because many development teams aren't used to monitoring applications, it can be easy to forget that while developing, testing

and deploying is the most time-intensive part of an application's life, the monitoring phase is in fact the longest. Effectively monitoring applications for errors and being able to quickly identify the source of problems is key to ensuring the application works for the end client on an everyday basis and that you are able to recover as quickly as possible from a crisis.

> "If you do not collect data in real time, as the functions are running, the data is lost."
> - Chase Douglas

Monitoring a serverless application is fundamentally different from monitoring a traditional application. First of all, serverless applications are generally built on microservices and have more complex architectures and dependencies. "if you have one thing that has to be monitored, that's a lot easier to do by hand or do through a manual, tedious process than if you have one hundred things or a thousand things," Goldstein says. There are both more functions to monitor—and more places where an error could occur—in serverless applications.

More importantly, though, is the ephemeral nature of serverless applications. A function runs and then goes away. There is no server to SSH into and retroactively collect data on an error after the function has run—if you do not collect data in real time, as the functions are running, the data is lost.

The ability to monitor a serverless application depends on setting up the logs during the initial process of defining the environment. It is theoretically possible to correctly configure all of the logs manually in AWS—however, doing so would require thousands of clicks. This highly manual way of setting up logs is time-consuming and so error-prone that it all but ensures that you won't have full coverage.

In addition, AWS's default log interface is incredibly challenging to navigate. "When you're just relying on the raw tools that AWS gives you, it is surprisingly difficult to even find the logs," Goldstein says. Using AWS's default logging tools, it could take hours and hundreds of clicks to find the information you need about a specific error—if the information exists at all. In a client-facing application—especially in a crisis situation—that isn't acceptable.

Choices for Moving to Serverless

Given the real challenges to successfully using serverless technology on client-facing applications, companies need tools to help define serverless environments, build serverless architectures, test and deploy serverless applications and monitor those applications once

they are live. One possibility is for each company to develop its own tools to manage the serverless application lifecycle. This requires a near-bottomless engineering budget and a deep operations bench. It also contradicts the lean and fast ethos of serverless development.

The second option is to rely on third-party tools like Stackery to automate environment management, architecture creation, deployment and monitoring.

HOW AUTOMATED ARCHITECTURE DEVELOPMENT AND DEPLOYMENT HELPS THROUGHOUT THE APPLICATION LIFECYCLE

Using tooling to manage your architecture development and deployment allows developers to create serverless architectures quickly, to make changes that are repeatable across environments and to deploy applications easily and without errors.

Development

Once a business logic for a new application has been established and a team has been assembled, an automated architecture development tool lets engineers quickly take their conceptual ideas for an effective application architecture from the whiteboard to reality in Amazon Web Services.

"You are able to visually see what you are trying to create in serverless," says Patrick Steger. "You can visually see all of your Lambdas." Instead of a list of code, you have an intuitive, visual interface that makes it easy to understand the connections between microservices.

The visual architecture design almost eliminates the learning curve involved in setting up a serverless environment. It's easy to explain to non-engineers in management how the application works; it's also simple to onboard new team members.

> "Tooling that automatically creates your serverless architecture dramatically reduces that amount of time it takes to create a serverless application; more importantly, it reduces the probability of errors in the logging and permissions configurations."
> — Sam Goldstein

Automated architecture tools let engineers quickly draw out their architecture design and connect services with a drag-and-drop interface while the tool generates the JSON code that creates the architecture in CloudFormation. The tool automatically configures the logs, configures IAM rules and correctly packages dependencies.

Tooling that automatically creates your serverless architecture dramatically reduces that amount of time it takes to create a serverless application; more importantly, it reduces the probability of errors in the logging and permissions configurations. It also helps combat what Steger calls "cowboy coding,"—sloppy code that doesn't adhere to best practices. Cowboy coding is fine for hobby projects developed by individuals, but can be fatal to business applications developed by teams that are part of a web of dependencies.

> "Using an automated architecture design tool ensures that you are collecting all of the information you need to troubleshoot the application as part of both mundane maintenance and to recover from a failure. " - Chase Douglas

"How do we actually make sure that a team of 10, 20, 50, 100 or a thousand engineers can actually be effective in shipping applications?" asks Douglas. The answer, he says, is with effective serverless tools.

Deployment

Instead of an error-prone, multi-step deployment process, engineers who use an automated architecture and deployment tool will be able to deploy their application to either a test environment or production environment in one click. There is no need to manually package hundreds of functions or for engineers to spend hours going through a manual deployment checklist.

Testing

One-click deployment is essential to fully testing a serverless application in a test environment for the following reasons:

- *If each microservice requires hundreds of human steps to deploy, the cost in terms of human resources to deploying that microservice twice—once to the test environment, once to the production environment—will simply be too great for most companies.*
- *Accurate testing depends on the ability to deploy the application in exactly the same way to the test environment and production environment. Any deployment process that involves hundreds of human steps is unlikely to be done exactly the same way twice, which renders the test environment useless.*

The ability to automatically manage deployment is essential to accurately testing your serverless applications—which is, in turn, essential to creating applications that perform reliably for end users.

Monitoring

Because serverless applications' performance and error data must be collected in real time, successful monitoring strategies require setting up the logs correctly at the time the architecture is defined. "In theory, you could have your architecture up and running in Amazon Web Services and you could go through a thousand clicks in their console to get all that stuff turned on… but with that type of process, especially when there's multiple people involved, you're not going to have comprehensive coverage," explains Goldstein, from Stackery.

Using an automated architecture design tool ensures that you are collecting all of the information you need to troubleshoot the application as part of both mundane maintenance and to recover from a failure.

If you don't have a way to quickly navigate to the relevant log, however, engineers could find themselves bogged down with information and unable to find the applicable log. By default, each instance of Lambda will create a separate log, creating a massive amount of data that is hard to make sense of manually.

Using a tool with an intuitive interface that both correctly configures the logs at the beginning of the application's life and organizes and curates the logs for easy, intuitive access at any time can mean the difference between fifteen minutes of downtime and a day of downtime.

> "With an automated architecture development and deployment tool, IT departments at an enterprise level can expand the use of serverless into public-facing applications and start taking advantage of not just serverless's reduced computing costs but also the quick time-to-market for serverless applications."
> — Chase Douglas

Monitoring isn't just about recovering from a crisis—it's also about peering into the application to see how it is running and where inefficiencies could be fixed. Without additional tooling, applications running on Lambda in Amazon Web Services are fairly opaque. Engineers aren't able to look into the application and analyze how it's performing in the real world. With additional tooling, however, developers can sort through curated logs and dive deeper into specific performance measures like how many times particular functions were run and if there were any errors generated.

> "Any process that requires a human to take tens–or hundreds–of steps is highly susceptible to errors." - Chase Douglas

This easily accessible feedback leads to better-quality applications and engineering teams that are more skilled and more comfortable creating serverless applications.

WHAT TO EXPECT WITH AUTOMATED ARCHITECTURE DEVELOPMENT AND DEPLOYMENT

Ready to try an automated architecture development and deployment tool for serverless environments?

Here's what you can expect:

Find the confidence to try serverless on public-facing applications. Many businesses have dipped their toes into serverless technology by migrating jobs and other internal, way-back-end functions to serverless. With an automated architecture development and deployment tool, IT departments at an enterprise level can expand the use of serverless into public-facing applications and start taking advantage of not just serverless's reduced computing costs but also the quick time-to-market for serverless applications.

Increase engineering team morale by focusing on creating great applications and eliminating tedium. Moving to a serverless environment eliminates the need for internal engineering teams to handle server management. Without an automated architecture design and deployment tool, however, engineering teams could find themselves stuck with a manual and onerous deployment process that take just as much time as server management. With a third-party architecture design tool, engineering teams can focus on identifying creative technological solutions to business problems.

Go from idea to launch as quickly as possible. Between the time saved by outsourcing server management and the time saved by using an architecture design and deployment tool, engineering teams will be able to move from idea to proof of concept to launch in a fraction of the time it would take in a traditional computing environment as well as faster than if they had to either manually design and deploy their serverless applications or build proprietary tools from scratch.

Avoid the human errors that plague all manual deployments. Any process that requires a human to take tens–or hundreds–of steps is highly susceptible to errors. Automating your architecture design, your log and permissions configuration and your deployment process dramatically reduces the likelihood of human error–and makes it much easier to locate the error source when there is a problem.

Make collaboration on serverless projects smooth, for experienced and newbie serverless developers. Serverless teams are generally sleek, but they're still teams. Using a tool with an intuitive interface–and automatically configured permissions–makes it easier to

onboard new team members, cuts down on “cowboy coding” and ensures that team members aren’t changing things that should be left alone.

Automated development and deployment tools make it possible for businesses to take full advantage of serverless technology’s reduced costs and potential for quick application launches—without sacrificing the ability to test, monitor and recover from crisis.

ABOUT STACKERY

The serverless architecture movement is transforming the ways modern organizations build applications and manage infrastructure. As early users of AWS Lambda, Stackery Founders Chase Douglas and Nate Taggart found themselves in need of a solution to the operational challenges presented by this technology. Having worked together as early employees of New Relic, Nate and Chase took their experience building for the developer and operations ecosystem and, with the early backing of Techstars Seattle, went on to launch Stackery to bring serverless technology mainstream.