STACKERY

Leveraging the Microservices Transition with Serverless

EXECUTIVE SUMMARY

As managing legacy monolith applications becomes increasingly cumbersome, many companies are investigating how integrating microservices into their application architecture can solve problems related to maintaining and updating the application, safely adding new features, managing scaleability and onboarding new developers. Microservices can solve a lot of pain points caused by a monolithic architecture, but they also create some challenges. Serverless technology allows companies to get the most benefits out of the transition to microservices while automatically solving many of the problems microservices create. The end result is that engineering teams spend more time developing unique solutions to business problems rather than managing servers, integrations or infrastructures.

WHAT MOTIVATES THE TRANSITION TO MICROSERVICES?

At an enterprise level, the transition to microservices is often motivated by increasing difficulty managing the legacy monolith. The most common frustrations that cause IT departments to start transitioning away from the monolith into a service oriented architecture are the following:

Maintenance and the Monolith

At one point, the legacy monolith was a sleek, state-of-the-art application, presumably written following best practices of the time. But as the years have gone by, most monoliths have been altered and added to; now there are millions of lines of code. Often no one fully understands the complex relationships between different parts of the code base, making it difficult to predict if a change to one part of the application will interact with seemingly separate functions in unexpected ways.

This unpredictability makes it complex, slow and risky to build new features or services and successfully integrate them into the monolith. In addition, no maintenance task is simple on a million-line codebase. Relatively straightforward updates like upgrading library versions are challenging and time-consuming.

Managing Team Members

Every new developer who starts to work on the monolith has to be familiar not only with the programing language in which it was written, but also with company-specific idiosyncrasies. A developer joining a team dedicated to authentication would still need to understand the entire monolith because anything he or she wrote could interact with the entire code base.

In practice, this means it could be months before even the most experienced new hire could function independently. This severely hampers team flexibility and the velocity at which a company can grow.

When every engineer on a large team is working on and making changes to the same monolith, there is a sense of shared ownership—in the sense that no one feels personally responsible for the health of the entire application. In practice, this often leads to a buildup of technical debt that can become overwhelming.

Monoliths are Difficult and Inefficient to Scale

There is no way to scale a single component of a monolithic application, so increased usage of one individual feature requires scaling the entire monolith. Computing resources have to be provisioned to account for peak demand of the monolith's most popular feature. This leads to significant waste of computing resources—which translates to wasted money.

Moving to a microservices architecture can help with all of these pain points. Microservices can be updated independently of each other, and there's less risk that a new service will break the entire application. The modular nature of a microservices architecture also makes it much easier to onboard new team members—there's no need for developers responsible for authentication to understand the shopping cart code. Microservices also allow for individual scaling, cutting down on wasted computing resources.

Although most companies start moving towards a service oriented architecture as a result of frustrations with the monolith, the benefits of microservices go beyond reducing pain points. Microservices generally allow teams greater flexibility, better security, greater ease in terms of onboarding new team members and in general a more agile engineering department that is better able to focus on the company's business logic. When microservices are paired with a serverless environment, these advantages are magnified while many of the downsides of a microservice-based architecture are mitigated.

> "Serverless solves a lot of the problems out-of-the-box that you need to solve for microservices." — Sam Goldstein

HOW CAN SERVERLESS HELP YOU LEVERAGE MICROSERVICES?

Moving to microservices will not decrease the complexity of your application; it will shift the complexity. Just like a tangled monolith with hundreds of features throughout millions of lines of code is hard to manage, hundreds of interdependent microservices also create substantial challenges. The easiest way to solve those problems is by running your service oriented architecture on serverless. "Serverless solves a lot of the problems out-of-the-box that you need to solve for microservices," says Sam Goldstein, Vice President of Engineering.

Managing Integrations

One of the biggest challenges with microservice-based architectures is managing the communication and integrations between all of the difference microservices. “You need something that can help you manage a lot of different small pieces that interact in sometimes very complex ways,” Goldstein explains. This can be done through container orchestration or by running the microservices on top of virtual machines, but either of those solutions require a highly-skilled team to manage the underlying infrastructure.

“Serverless is very well-suited to APIs and message passing,” says Nate Taggart, CEO of Stackery. “Serverless provides an architecture that's really designed for microservices. You could build the exact same thing, still using APIs, still using message passing, and then run it on your own server or run it on a cluster. That doesn't really change it from the microservices standpoint, but you have the added responsibility of managing all of these different services.”

Using appropriate serverless tooling can also help leverage the benefits of serverless even further. Operational tools such as Stackery provide real-time visibility into how microservices interact as they respond to user requests. These operational tools make it easier to both design the architecture in the first instance and to make changes and correct bugs after deployment.

Serverless Manages the Infrastructure

If you’re running microservices using containers or on top of virtual machines, you’ll need to have an internal team dedicated to either container orchestration

The Strangler Approach and Serverless

“‘We have a microservices architecture could mean anything from we have a monolithic application and one service to we have 200 services and no monolith in the middle,’” explains Nate Taggart, CEO of Stackery. Just as the transition to microservices is generally a gradual transition, with more and more services being broken off the monolith until the monolith disappears, the transition to serverless generally follows a similar pattern. In addition, serverless can help make the strangler approach to microservices practical by creating serverless API layers, abstracting away the old, fragile API. Using serverless operational tools like Stackery allows you to strangle your monolith while easily monitoring and managing operations.

or provisioning virtual machines. You'll need to provision infrastructure and manage load, scale and availability internally. The level of talent required to do this successfully is scarce—even if you can afford it, engineers with the necessary skills might prefer to work for a 'cooler,' more tech-focused company. In addition, investing in infrastructure management offers little competitive advantage for most companies. Unless your company has a business reason to be the best in its industry at container orchestration or virtual machine management, running microservices on either system involves wasting a considerable amount of resources. "At most companies, it's not like they will get a competitive advantage from building out a big team of highly-paid specialists so that their dev team can succeed with microservices," explains Goldstein. "It may be necessary, but it doesn't actually provide any benefit. With serverless, you're outsourcing that piece, so instead of becoming a specialist in container orchestration, you can focus on building technology that does give you a competitive advantage."

> "Serverless provides an architecture that's really designed for microservices. You could build the exact same thing, still using APIs, still using message passing, and then run it on your own server or run it on a cluster. That doesn't really change it from the microservices standpoint, but you have the added responsibility of managing all of these different services." — Nate Taggart

Using a serverless environment for microservices eliminates the need for in-house infrastructure management, freeing up engineering resources for other tasks. It also gives companies a way to leverage Amazon's infrastructure management capabilities, which are better than what the vast majority of companies are able to produce in-house.

Automatic Individual Scaling

One major advantage of moving to a microservice architecture is the ability to scale one individual component without scaling the entire system. Serverless is the only way to take full advantage of individual scaling. While individual scaling is technically possible in a container system, there is often a minutes-long lag time as individual services scale and more containers are provisioned—and getting individual service scaling to work correctly in a container system is challenging from an engineering perspective.

In a serverless environment, scaling is automatic and handled by the serverless provider. Microservices can scale up or down in seconds. Just as importantly, this rapid individual scaling is an out-of-the-box feature, so getting it to work requires no additional engineering investment other than moving to serverless. “The ability to scale on demand is very useful, especially since you don't need to solve these incredibly difficult engineering challenges to do so,” Goldstein says. “You can just say well, let's deploy our stuff to Lambda and it just works.”

Pay-Per-Use Cost Model

Serverless’s unique cost model allows companies to pay for the computing resources they actually use instead of provisioning and paying for estimated peak usage. When combined with microservices’ individual scaling, the pay-per-use cost model can lead to substantial cost reductions.

“Pretty much any website or system is going to have highs and lows,” Goldstein explains. “Only paying for what you’re using means you don’t have to provision for peak capacity, and have much of your capacity sitting idle most of the time.”

Because both individual scaling and pay-per-use billing are included out-of-the-box in any serverless environment, companies running microservices on serverless take advantage of this cost structure automatically, without any additional work from the engineering team.

Track Usage and Costs for Individual Features

Serverless’s automatic individual scaling and pay-per-use cost structure make it possible, with the correct tools, to get unprecedented visibility into the computing costs associated with running individual components of an application. This allows leadership teams to track the costs associated with each feature and use that information to identify ways to optimize the company’s cost structures.

“With serverless, you have the potential to do a much better job tracking how the product cost breakout compares against the revenue it generates,” Goldstein explains.

While this granular cost visibility isn’t an out-of-the-box feature in serverless environments, running microservices on serverless with the appropriate tools facilitates this level of health tracking and cost visibility in a way that isn’t possible with any other architecture or environment set-up.

Faster Time-to-Market

The time-to-market advantage of microservices is perhaps the leading reason that companies start moving away from their legacy monolith—but using serverless is the only way to actually deploy new microservices substantially faster than new features on a monolith. “If your time-to-market involves setting up an enterprise container orchestration platform, you may not actually get to market quickly,” Goldstein says.

Outsourcing the server provisioning and infrastructure management to the cloud service provider by deploying on serverless cuts the deployment time for an individual microservice dramatically, from potentially months to as little as hours.

Focus on the Business Logic

The goal for any engineering team should be to focus as much as possible on creating unique solutions to business problems and opportunities. Transitioning to microservices is a way to reduce the mundane maintenance required of engineering teams working on monoliths, but a microservice-based architecture comes with its own set of tasks required to keep the application working smoothly. Using serverless transfers responsibility for most of these background tasks to the cloud provider, freeing up in-house engineers to work on meeting customers’ needs.

OVERCOMING COMMON BARRIERS TO USING SERVERLESS FOR MICROSERVICES

Although the payoff is worth it, there are legitimate barriers to a serverless transition. Some of these barriers are real but overcome-able while some barriers are “fictional,” or rooted more in fear about new technologies and processes than in facts about using serverless environments.

Here are some common obstacles to moving microservices to serverless—and how they can be mitigated:

Microservice Architectures have Complex Dependencies… and Failures

An architecture based on microservices is just as complex—or more so—than a monolith, but the complexity is expressed differently. A complex web of interdependent microservices means that failures, when they happen, can be both harder to diagnose and more serious than failures in a monolith. “When you’re transitioning from monolith to microservice, you’re trading high probability low impact risk for low probability, potentially higher impact risk,” explains Nate Taggart, Stackery’s CEO. This is actually an argument in favor of moving microservices to serverless: In serverless, you’re outsourcing a lot of that risk to Amazon.

“If your time-to-market involves setting up an enterprise container orchestration platform, you may not actually get to market quickly.” — Sam Goldstein

However, without appropriate tooling, serverless environments can be very opaque, and diagnosing a problem is near-impossible if best practices weren't followed at the time of deployment. This can make companies uncomfortable with moving business-crucial parts of their application to serverless.

Correct operational tooling solves this problem by ensuring that best practices are consistently followed throughout the development and deployment process. With the correct tools, preventing, diagnosing and fixing failures in a microservices architecture is easier in serverless than in any other kind of computing environment.

Serverless vs Containers

Containers and serverless are often presented as competing technologies, but in reality there are situations in which containers are more appropriate and other situations where serverless is the better choice.

Serverless is best for:

- Scaling individual microservices
- Removing the need to manage infrastructure
- Managing complex microservice integrations
- Quick time-to-market for new features
- Visibility into the costs and resources used by each service

Containers are best for:

- Processes with a long run-time
- Situations where legacy applications need to be "lifted and shifted" quickly, since it is easier to transfer most legacy applications to containers than to serverless
- Any situations where full control of the environment is essential

Because containers require substantial, continuing investment in orchestration, they should only be used for the services that aren't practical to run on serverless.

Microservices Require Maintaining and Updating 100s of Services

When handling a single, monolithic application, it is possible for engineers to handle updates, changes and operational concerns manually. In an architecture with hundreds of microservices, manually making changes is impossible.

Microservice architectures therefore rely heavily on automation and tooling. Serverless does not provide any out-of-the-box solutions for managing automation, and as a relatively new technology, third-party tools to provide the automation that service oriented architectures rely on is relatively immature.

Managing automation throughout the application lifecycle is one of the core benefits of using Stackery's Operations Console, and can solve many of the management problems facing complex microservices architectures running on serverless.

Parts of Your Application Might Be Incompatible with Serverless

Just as there might be some parts of your application that shouldn't be broken apart into microservices, there might be certain functions that don't make sense in serverless.

An obvious example relates to run-times. In general, serverless functions can't run for longer than five minutes, so long-running parts of your application wouldn't be appropriate for a serverless environment.

> "Moving to serverless is a relatively small shift, in terms of learning curves, compared to the changes required in the transition to microservices. " — Nate Taggart

Serverless, like microservices, is not an all-or-nothing proposition. "As you're making this transition, you are kind of opportunistically picking off areas where you can break the service out from your code base," Taggart explains. As services are broken off from the monolith, they can be transitioned into serverless if appropriate—but stay in a traditional cloud environment if there are compelling reasons against using serverless.

Moving to Serverless Involves Another Learning Curve for your Team

If you're in the midst of transitioning from a monolithic application to a service oriented architecture, your engineering team is already learning new skills, taking on new roles and adjusting to a new way to work together. Transferring some or all of the microservices to serverless requires that engineers learn new skill sets, too—but the learning curve is relatively minor compared with the changes involved in going from a monolith to microservices. In addition, engineering teams have to be adapting and changing continually—learning new sets of best practices, adopting better security practices and generally keeping pace with progress. The new skills required to move microservices to a serverless environment shouldn't be a major impediment to serverless adoption for most teams.

In addition, using intuitive operational tooling as part of the transition to serverless can dramatically reduce the time it takes for engineers to become comfortable running microservices on serverless while also reducing the risk of errors during the learning process.

Moving to a microservices architecture is an institutional investment in reducing the learning curve for all new team members in the future. "You're already switching strategies from monolith to microservice and doing the legwork of having to re-architect," explains Taggart. "Moving to serverless is a relatively small shift, in terms of learning curves, compared to the changes required in the transition to microservices."

Concern about Lock-In

Although there is no more lock-in associated with serverless environments than there is in using any other cloud provider or equipment made by a third party, the opacity of serverless environments—and the fact that running on a local machine isn't possible—make some teams concerned that moving from one serverless provider or back to a traditional cloud environment could be extremely difficult.

While using serverless environments does involve more of a commitment to a specific vendor than running an open-source container on virtual machines, microservices running on serverless are written using standard programming languages and could be migrated from one provider to another (or back to a traditional cloud environment) relatively easily. So while lock-in gets a significant amount of press, in reality it should not prevent any company from adopting serverless.

WHAT MICROSERVICES ON SERVERLESS LOOKS LIKE

Although the payoff is worth it, there are legitimate barriers to a serverless transition. Some of these barriers are real but overcome-able while some barriers are "fictional," or rooted more in fear about new technologies and processes than in facts about using serverless environments.

Here's what to expect as you move your microservices into a serverless environment:

Decreasing Environment / Server Costs

As more and more of your application is run through individually-scaleable microservices in a serverless environment, you will increasingly be paying only for the computing resources that the application uses instead of for the maximum amount of computing resources the entire application would need during a spike in demand. This cost model leads to substantial savings for most companies. The savings increase as more of the application is run using microservices on serverless.

Dramatic Drop in Time Spent Provisioning Servers, Managing Integrations and Infrastructure

If you're running a microservices architecture on an in-house data center or in a cloud environment, determining the computing resources for each individual service and provisioning the required servers can eat up a substantial amount of time. This delays new feature deployment and also is an inefficient use of your engineering team's time. Likewise, a microservices architecture run on containers or virtual machines will require a team of engineers to manage integrations and the underlying infrastructure. In serverless, server provisioning, integrations and infrastructure is handled automatically and immediately by the cloud provider. Running microservices on serverless can free up entire teams to work on other projects. Using appropriate tooling can reduce the time spent managing integrations and infrastructure even further.

Quick Releases

Eliminating the need to manage servers provisioning, integrations and infrastructure in-house means that new microservices deployed to a serverless environment get to the market much faster than the same microservice would if deployed using containers, virtual machines or other technology. Serverless enables companies to get new features to their customers in a matter of hours instead of months. "It's a time to market benefit that really drives people to serverless," Goldstein explains.

Focused Engineering Teams

Engineers are among the highest-paid employees at most companies, and their skills are ideally used to create unique solutions to business problems. In reality, most engineering teams have to spend a substantial amount of time on tasks like infrastructure management and server provisioning that don't give the company any competitive advantage.

A microservices architecture run on serverless allows engineering teams the maximum amount of focus on the company's core business logic by outsourcing server provisioning and infrastructure management. Using available serverless tools like Stackery further simplifies the complexity of a microservices architecture and eliminates the need to build operational tooling in-house.

The result: Engineering teams are able to spend their time creating new functions and features that meet customers' needs and provide a competitive advantage—and they are able to do so dramatically faster than possible when working with either a monolith or microservices in a non-serverless environment.

GET THE MOST OUT OF MICROSERVICES AND SERVERLESS WITH STACKERY

One of the biggest barriers to widespread serverless adoption is the gap in operational tooling for serverless environments. Tooling is essential when handling complex microservice architectures with hundreds of interconnected services, and visibility and monitoring capabilities are key to successfully launching and running any business-crucial application. Stackery's intuitive serverless operations console provides the automation and visibility needed to successfully run complex architectures on serverless. "We build out the best practices under the covers, and we do it in the cloud native way," explains Nate Taggart, Stackery's CEO.

Intuitively map, create and change the complex dependencies in your microservices architecture. Stackery's infrastructure provisioning interface allows you to easily visualize the complex relationships in your architecture and manage the connections between services with a drag-and-drop interface.

Automate your deployment process. Managing hundreds of microservices without automated deployment processes is asking for trouble. Stackery provides automatic deployment management so you can ensure that all microservices are packaged and deployed according to best practices. Stackery also dramatically reduces the amount of time spent on deployments.

Access intuitively curated logs, metrics and health tracking information. Stackery provides detailed visibility into the performance of both individual services and the application as a whole, giving you a level of insight that is not otherwise possible.

Get full diagnostic details on all errors. Stackery automatically wraps code in Try/Catch logic, ensuring that you have detailed error and error trace information collected in real time. Stackery's error monitoring ensures that developers get a complete stack trace for any errors and are able to pinpoint the problem microservice (s) in seconds.

Speed up the learning curve. Stackery's operations console makes it easier for developers new to serverless to get started quickly while enforcing best practices and managing permissions, so if a mistake happens it doesn't cause catastrophic failures.

Focus on Your Core Business. Using Stackery to create, monitor and manage microservice architectures on serverless means outsourcing as many tasks as possible, from server provisioning to infrastructure management to operational tooling. This gives your engineers the freedom to focus on the things that make your business unique.

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.