The Future is Serverless

When and How to Make the Switch

Kong

The Future is Serverless

When and How to Make the Switch

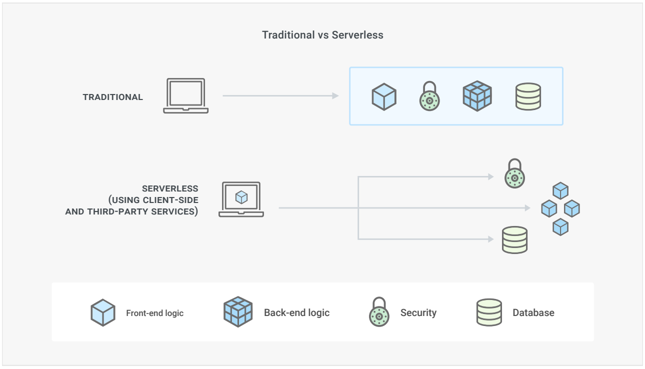
Content

The Future is Serverless: When and How to Make the Switch

You have known about the cloud for years. But as of late, you've probably been hearing more about Kubernetes and Docker. That said, serverless is emerging as a significant cloud architecture paradigm.

Serverless represents a dramatic shift in how we approach building applications in the cloud. The repercussions will be felt not just at the engineering level but higher up the application stack at the business level. Serverless computing can dramatically change the way you manage your cloud infrastructure and build your systems. Instead of building your entire system or microservice, you can deploy a few functions at a time. You can just focus on the code and know that your functions will scale with your application—without you having to manage the infrastructure.

We'll start by defining serverless, review use cases and how to determine if serverless is a good fit for use in your stack, how to secure and manage it at scale, and lastly, how this change may affect your business in the future.


Traditional vs Serverless
TRADITIONAL
SERVERLESS (USING CLIENT-SIDE AND THIRD-PARTY SERVICES)
Front-end logic
Back-end logic
Security
Database

What is Serverless?

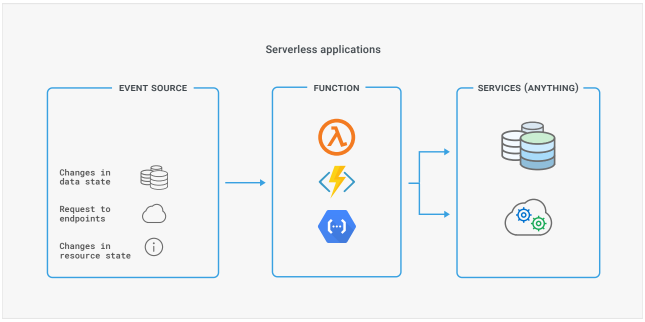
So what exactly is serverless? To start, yes, there are still servers involved. However, what makes something serverless is the fact that while those servers exist they are irrelevant to your application architecture. As a result, your engineering team only needs to focus on the application code.

There are many different and competing definitions of serverless floating around. So, let's focus on some common characteristics for the purpose of this article. Serverless is a cloud architecture that generally adheres to the following characteristics

- No servers, virtual machines, or containers to manage

Fig 1: Weston, Robin. Serverless Architectures and Continuous Delivery

- A reliance on managed services; either at the cloud provider, third party, or SaaS level
- Event-driven architecture resulting in application code only running when triggered
- A cloud provider billing model where you're only charged for what your application uses



Serverless computing changes the shared responsibility model. Instead of you managing the OS/runtime, the platform as a service (PaaS) provider now manages it for you. This allows your engineers to focus on the code.

These are some common characteristics of serverless at a high level to help us to better understand what we're talking about. Now let's dive deeper.

Fig 1: Amazon Web Services

FaaS

When people talk about serverless, they most often think about functions as a service, or “FaaS”. This is because FaaS is at the heart of most, but not all, serverless applications. It’s also the most visible part of most cloud providers’ serverless offerings as AWS promotes Lambda, Google promotes Cloud Functions, and Microsoft promotes Azure Functions.

Serverless functions are not full applications. Take an application, e.g. a backend REST API, and break it down into smaller pieces. Each endpoint on that REST API becomes its own independent function that performs a specific narrow task and these functions are triggered to run only when it receives an HTTP request.

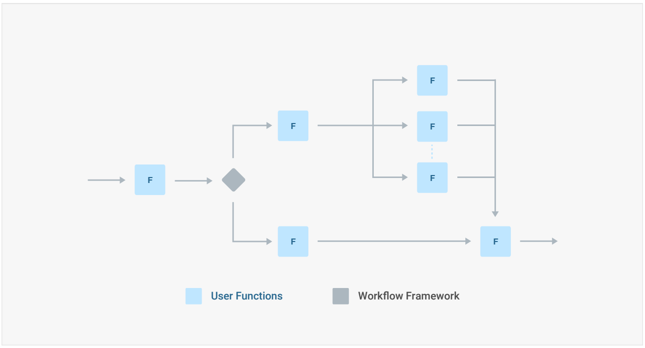
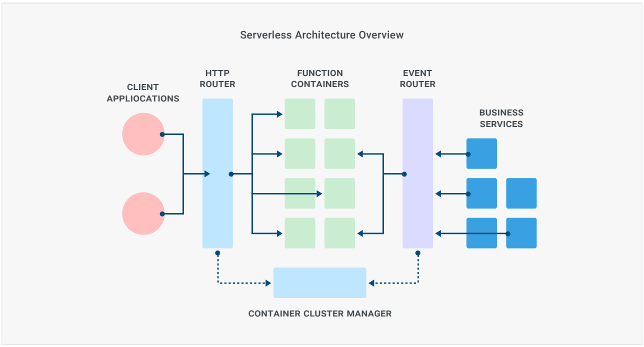


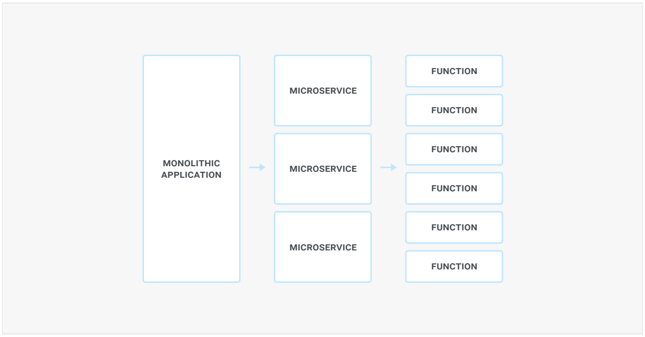
Fig 1: Fission.io. Serverless Workflows for Kubernetes.

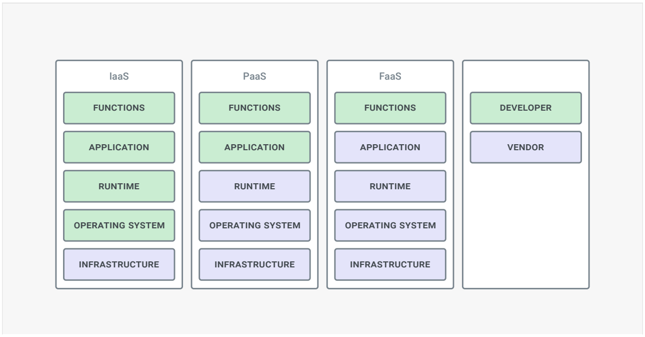
Rather than a single application that can create, get, update, and delete records in a database, you have independent functions that perform these tasks and can be managed entirely on their own. This means that they can be updated and deployed independent of one another. Plus, by decoupling these functions you reduce the risk of introducing a bug in one function that prevents another function from performing.



Functions running on a FaaS platform are expected to be both short running and ephemeral. That means they should only run for up to a few minutes, enough time to complete their narrowly-focused task. Ephemeral means they do not maintain the application state in between invocations; rather, any application state should be managed using an external datastore.

Fig 1: Gunaratne Imesh. "Adapting Serverless Architecture." DZone Guide to Cloud, Volume IV.





There are a variety of FaaS platforms out there, both publicly and privately hosted. In the public cloud space AWS, Google, and Microsoft provide FaaS services along with additional managed cloud services to build serverless applications.

Privately-hosted serverless platforms include Knative and Apache OpenWhisk. While Knative is Kubernetes dependent, OpenWhisk can run

Fig 1: Cloudflare. "What is Function as a Service."
Fig 2: Assist: "Pros and Cons of Serverless Computing. FaaS comparison"

on cloud management platforms including Kubernetes, Mesos, and Docker Compose. These private serverless platforms allow you to leverage serverless architecture on top of your existing cloud management infrastructure.

Other types of serverless platforms

FaaS applications aren't the only type of cloud applications that can be considered serverless. There are many other serverless computing services available including object stores, NoSQL databases, and notification services. While each of the serverless computing offerings run on infrastructure, you as the user of the service don't need to worry about managing the infra-

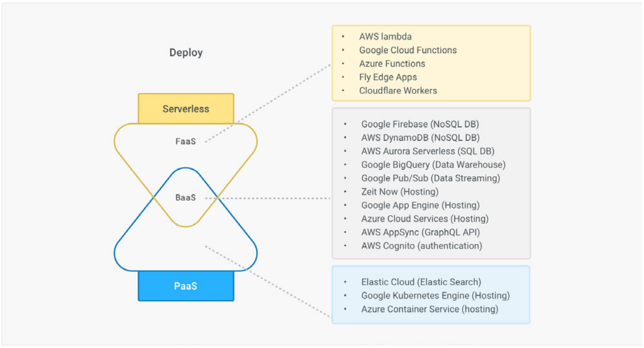

Deploy
Serverless
FaaS
BaaS
PaaS
AWS lambda
Google Cloud Functions
Azure Functions
Fly Edge Apps
Cloudflare Workers
Google Firebase (NoSQL DB)
AWS DynamoDB (NoSQL DB)
AWS Aurora Serverless (SQL DB)
Google BigQuery (Data Warehouse)
Google Pub/Sub (Data Streaming)
Zeit Now (Hosting)
Google App Engine (Hosting)
Azure Cloud Services (Hosting)
AWS AppSync (GraphQL API)
AWS Cognito (authentication)
Elastic Cloud (Elastic Search)
Google Kubernetes Engine (Hosting)
Azure Container Service (hosting)

Fig 1: Joan, Naame Seraphine. "All You Need to Know About SAAS Architecture."

structure; the PaaS provider takes care of this.

Additionally, static websites can be built using a serverless approach. Consider a static website built with HTML, CSS, client-side JavaScript, and images where you're only charged for the number of site requests made in a month. Now compare this with a similar site running on hosts where you're charged by the hour, whether or not you are actively serving up content. Looking further out, as more developers move towards managed GraphQL services, we will likely see an increase in serverless dynamic web applications lacking a FaaS component.

Why Serverless?

Why would you adopt a serverless architecture? To start, it decreases the complexity of operating cloud infrastructure for an individual engineer. As a result, an engineer spends less time focusing on infrastructure and more time on their code and the problem their code is expected to solve. Furthermore, the reduced cloud complexity means less friction between developers and operations. This results in faster feature development and faster delivery to production.

Cost implications are another reason for serverless adoption. Why pay for a cloud service when you're not actively using them? By only paying for cloud resources that are actively being used, an organization can cut costs. This is what makes infrequently-used applications in your environment a perfect candidate for serverless. However, not all applications are created equal and some will see better cost benefits than others.

Finally, and most importantly, an organization can focus on their core competency. Today, every company is a technology company. Every company has a large technical investment required to keep their organization running. Going serverless allows an organization to spend less time and resources on operating applications. The newly-found time and resources can be redirected towards delivering value to customers. What makes your organization unique isn't its cloud infrastructure, it's the features and values you provide to waiting users.

Pros / Cons

As with all technology, there are pros and cons. Here are the main points you should consider before adopting a serverless solution:

Pros

- Highly scalable: The first pro of serverless is its scalability. Your serverless platform is responsible for allocating and managing function instances to meet demand. Contrast that with microservices on a traditional server architecture where you're responsible for scaling hosts either vertically or horizontally to meet demand. This requires manual intervention or autoscaling rules that can be slow to execute.
- Reduced cloud operations work: Although serverless does not completely eliminate this, it reduces cloud infrastructure operations issues. How much time does a developer spend waiting for a host to deploy their new code? With serverless, deploying both the application code and the supporting infrastructure simultaneously only takes minutes. There's also less cloud infrastructure to go wrong and impact the application code. How many times does a host issues with CPU, memory, or disk cause application issues? These are no longer issues for you.
- Agility and focus: Finally, serverless lets application developers focus on the problem their code attempts to solve without being distracted by cloud infrastructure.

Plus, the ability to rapidly deploy means a team can ship and measure the success of more features. A team adopting serverless should be able to ship more features and accurately measure whether or not they have shipped the right features.

Cons

- Vendor lock-in: If your organization is concerned with vendor lock-in, then serverless is not the right choice for you. The deeper you go into adopting a cloud provider's services, the more value you obtain from it. If you're worried about being too reliant on a cloud provider and want greater nimbleness to switch providers, serverless isn't the best choice for you.
- Engineering practices still emerging: Serverless engineering is still emerging. That means the patterns, processes, and tools are still in development. Additionally, finding people readily skilled in serverless engineering is not easy. You will need to take the time to develop your organization's best practices and train resources.
- Not right for every workload: Serverless is not right for every workload, particularly long running workloads. Serverless is designed for short running ephemeral tasks.

> If your workload requires extensive execution time or needs to keep track of state on disk in between executions, then it's not right for serverless. But, that's an issue a little, or a lot, of refactoring might solve.

Neutral

Cost: Several factors can impact the cost of serverless services including the workload, the type of service being used, and platform pricing models. Not all serverless workloads involve the same costs. That means some applications will benefit significantly over running them on servers; others may be more expensive.

When Should You Use Serverless?

When is it appropriate to use serverless solutions? And more specifically, when should you use serverless functions? Some serverless services like S3 and DynamoDB do not have any duration or usages limits. This means you can store as much data as you like in an S3 bucket.

When you need to process compute workloads, functions like Amazon Lambda come with a few limits that you should be aware of. (You can find the complete list here: https://docs.aws.amazon.com/lambda/latest/dg/limits.html.)

1. The maximum size of the package you upload to AWS Lambda (currently this is 50 MB).
2. The amount of memory used by your Lambda function (the maximum limit is about 3 GB).
3. The duration of your function needs to complete within 15 minutes.

These limitations drive which use cases are best for serverless functions. Use a serverless function whenever you have a short-lived compute-based task to perform.

There are many different use cases where using serverless functions are a good choice:

- **Data processing** - Managing real-time data analysis (instead of an ETL job)
- **Chatbots** - Power your chatbot from a serverless function
- **Voice-enabled apps** - Like Alexa, use a Lambda function to power your voice UI
- **Automation** - Use Serverless functions to

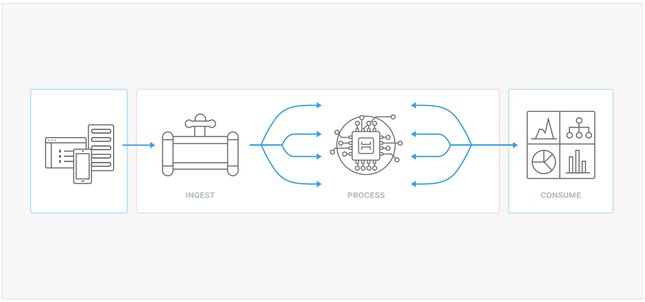
manage your cloud infrastructure, enforce policies, and so on
- **IoT** - Use serverless functions for server side IoT processing, since large networks of IoT devices can produce massive amounts of data.

Let's explore these use cases in more detail to get a better understanding. Here are several key use cases where serverless functions are a great fit:

Use Case 1: Processing data after it's uploaded to cloud storage

Serverless functions can be triggered in real-time to process data immediately after it's uploaded to an Object Store like S3, Cloud Storage or Azure Storage. For example, you've created an expense tracking mobile app. When your users take a picture of a receipt you want to store the image in the Object Store and then process the image. This is a great compute use case for a serverless function.

To implement this solution, configure the Object Store to invoke a specific serverless function whenever an object-create event is triggered in S3. You aren't limited to just working with an Object Store; you can use other Serverless Computing services from your function. In this example, we could process the image, store the image in a different Object Store bucket, and then update a row in a serverless database table.

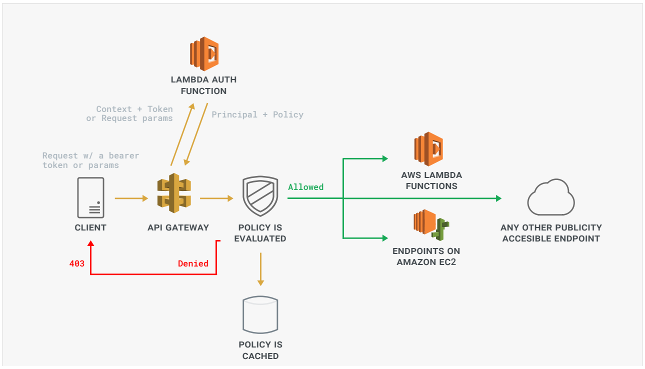


Use Case 2: Database access for client side apps

It's becoming more common to build apps with rich client side functionality such as mobile apps or single page web applications built on frameworks like React. For security reasons, they lack direct access to databases and you typically want a web service to authenticate and validate input.

Fig 1: Amazon Web Services

For example, users who are signed in want to see how many times their profile has been viewed and by whom. When a user clicks on their profile, we want to display their latest profile statistics. Your web application makes a GET request to a REST API Gateway which in turn invokes a serverless function. The serverless function makes a database call to get the user profile data, summarizes the data, and returns the result. Then, the web application displays the data. This a short-lived operation and doesn't require running a dedicated server.

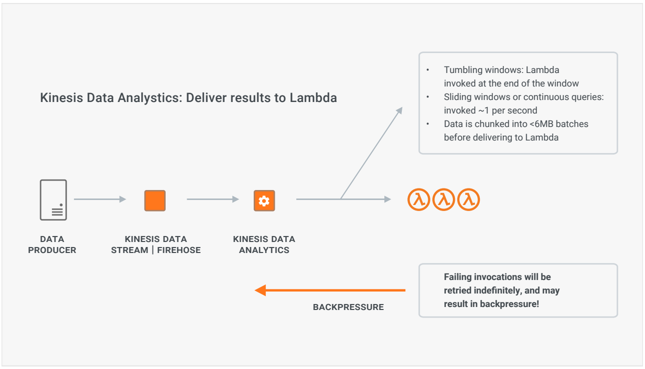


Use 3: Processing data streams

Serverless functions are great for processing streams of data that update on some regular basis. For example, a function could be used along with Apache Kafka or Amazon Kinesis

Fig 1: Munro, Jamie for Sky Betting & Gaming. "Optimising an AWS Microservice."

to aggregate data and store it in a database. In this use case, Kafka will store a data stream, and then a Lambda function will poll Kafka to consume data in batches of 50 items and store it in a database.


Kinesis Data Analytics: Deliver results to Lambda
Tumbling windows: Lambda invoked at the end of the window
Sliding windows or continuous queries: invoked ~1 per second
Data is chunked into <6MB batches before delivering to Lambda
DATA PRODUCER
KINESIS DATA STREAM | FIREHOSE
KINESIS DATA ANALYTICS
Failing invocations will be retried indefinitely, and may result in backpressure!
BACKPRESSURE

Use Case 4: Bursts of activity

If your application receives an unexpected burst of activity, it can be expensive to maintain extra server capacity, and even auto-scaling rules can be slow to execute.

For example, let’s say your backend service supports both a web site and mobile application. Today’s your lucky day! Your app is featured on the “Today” page of the AppStore and then featured on Hacker News. With this success comes an unexpected wave of activity.

Fig 1: Amazon Web Services

By using a Serverless design for your backend, your users would never know you weren't expecting them. Instead of your system failing because it can't handle the load, your serverless infrastructure scales to meet the need. Companies like The Financial Industry Regulatory Authority (FINRA) can process up to half a trillion serverless function calls a day. It's likely that any load your system receives—whether planned or unplanned—will be handled gracefully by a serverless architecture.

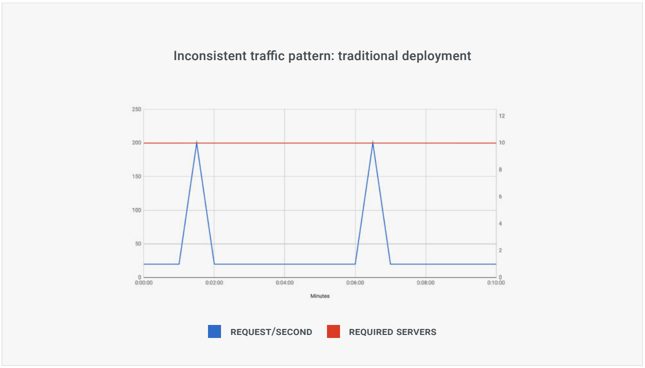


Fig 1: Roberts, Mike. "Serverless Architectures"

When to NOT Use a Serverless Function?

Using a serverless architecture doesn't give you control over the type of compute instances used. Depending on your system needs, the serverless function execution environment may not be appropriate.

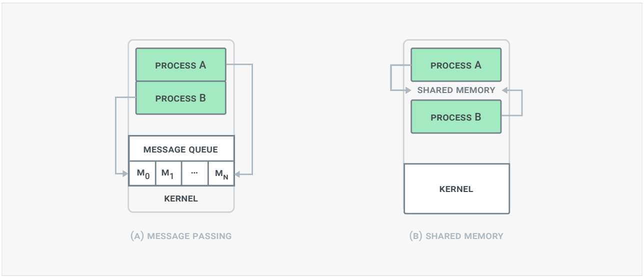
Here are several use cases where serverless functions are not ideal.

Use Case 1: Long running code

Serverless functions typically have a maximum execution time. Therefore, they are not ideal if your code runs continuously or needs to perform long computations. For example, your application has a scheduling algorithm that matches thousands of students and classes with teachers. The matching algorithm takes about an hour to complete; this code will not run successfully as a serverless function. You'll either need to change your implementation or deploy your application to a Compute resource since there's no way to change the serverless function execution time limit.

Use Case 2: Interprocess communication

Since serverless functions run in an isolated environment, there is no way for two functions to share memory. A scenario like this most likely comes from a multi-threaded implementation where each thread uses a shared memory region to pass data back and forth. That said, you could use a separate cache to store data. This cache can be shared between the functions or invocations of a function.



Use Case 3: Legacy systems

If you are moving your existing system to a PaaS, then chances are it's not designed to run in short bursts of activity; therefore, it's not appropriate for serverless functions. Since most enterprise systems are designed to start up and run forever, or until they stop or crash, a lift-

Fig 1: W Schools. "Interprocess Communication (ICP)"

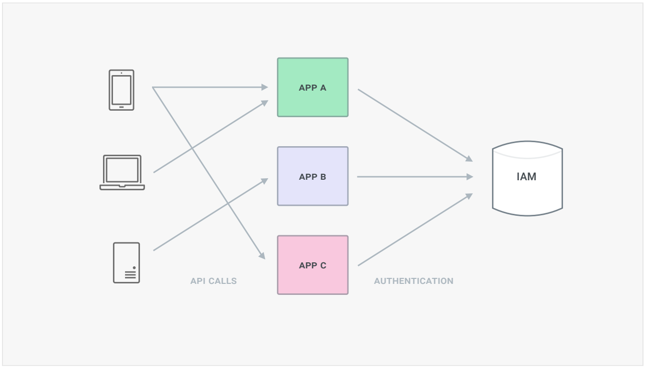
and-shift approach may not fit the serverless computing model. Once you systems move to the cloud, however, you can start moving portions of your application to serverless.

Managing Serverless Services

When considering a move to serverless functions, you should also look at how to effectively manage them. Since adopting serverless reduces your operational and management overhead, you need tools in place to ensure that functions run reliably, securely, and economically.

Authentication and Authorization

Application authentication and authorization involves the adoption of cloud provider services (e.g. AWS Cognito), third party SaaS services (e.g. Auth0), or even custom authorization functions running on the same functions as a service platform. Organizations typically depend on some sort of token or API key functionality to control public access. Compliance requirements may even dictate that employees be granted access to sensitive data on a limited basis. Relying on an IP whitelist or VPC connection may not offer the fine-grained control needed in large and diverse organizations.



Rate Limiting

The built-in scalability of serverless might make you think there isn't a need for rate limiting, but that's not true. Most platforms offer a usage-based payment model, so spikes in usage can lead to a large bill, whether intentional or not.

For each API request that comes in, your cloud provider is responsible for allocating a function instance to handle the request. With the scalability of serverless and the burden of meeting demand shifted to the cloud provider,

Fig 1: Optiv. "Five Security Best Practice for Serverless Applications."

why would API limiting be necessary? That's because on a practical level, cloud providers impose limits or quotas on services. API limiting is necessary so that overuse doesn't bump into these limits. What's more, these limits are often account wide. That means an overused API can affect the ability of other APIs to function successfully. Fortunately, the cloud providers have the ability to rate limit their API services or limit the concurrency of function invocations. For smarter rate limiting logic, you may want to consider using your own API Gateway like Kong for rate limiting.

Observability: Metrics, Monitoring, Logging, and Tracing

We mentioned before that serverless reduces but does not eliminate cloud operations work. Serverless, with its event-driven architecture, creates a degree of application complexity. When something goes wrong, where did it go wrong in the execution chain? Keeping serverless applications running means investing in the observability of your whole application stack. But it's not an increase in overall workload. The time not spent on managing hosts means more time proactively ensuring the reliability of your application. More time can create a better state of readiness and reliability than your applications today.

Security

Serverless security implications are similar to today's virtualized or containerized microservices. Application code and application dependency security are both highly important and the same practices carry over to serverless. But without servers there isn't any need to spend time on host-level security issues such as software patching and access controls. Instead, turn your attention toward securing cloud infrastructure and your function code to prevent attacks like remote code execution or SQL injection.

Using an API Gateway

API gateways like Kong can make it easier to manage your serverless functions. Rather than implement all these management features within each function, Kong allows you to put them in a centralized and easily managed location. It acts as proxy and accepts calls from clients, then passes them to serverless functions. Adding features like authentication, security, observability and more is as easy as adding a plugin from Kong Hub. Also, an API Gateway that is separate from your cloud provider allows you to manage functions in a multi-cloud environment.

This further reduces the amount of development effort required to publish new functions. The combination of not having to worry about server infrastructure or service management means that your developers can focus on business logic and accelerate your product's time-to-market.

How Serverless Will Change Your Business

Technology altering organizational processes and structures isn't new. Just look at public cloud adoption and its impact. The same will happen again with serverless. The competitive advantages will belong to companies that adopt these practices early on. Here are some ways the world is already beginning to change.

The Rise of DevOps

To start, organizations that go serverless will see a shift towards developers and operations working more closely in DevOps roles. By combining infrastructure and code together, software developers won't need operations engineers to get their code into production. Some organizations will see traditional operations engineers reskilling and becoming a part of development teams. They will handle the operational needs of their teams' services in this new position. This is a DevOps dream: a full cross-functional team involving all members of the software development and operations lifecycle.

More Focus on Business Outcomes

Engineering teams will become more focused on achieving organizational objectives. The focus, speed, and agility serverless provides means more emphasis on outcomes. Why? Because the team now has the time to do so. Struggling less with the upfront technical needs of their work, teams can deliver faster and spend their time measuring the success of what they've delivered. Picture a team that isn't only focused on what they've engineered, but whether their engineering is valuable and worthwhile.

Adopting a serverless architecture is not an easy task for companies with an extensive code base. Solutions like Kong make it easier to adopt and manage serverless functions by providing you with out-of-the-box support for authentication, rate limiting, and more. Check out our one-pager on how to Manage and secure your serverless functions with Kong.

Kong

[Konghq.com](https://konghq.com)

Kong Inc.
contact@konghq.com

251 Post St, 2nd Floor
San Francisco, CA
94108 USA