The 4 Top Serverless Challenges
The 4 Top Serverless Challenges

Thousands of companies have adopted serverless for greater parts of their technological stack. A 2018 report by New Stack found that more than 75% of companies use or plan to use this latest technology. And that number is going up because it's easier, cheaper, and more efficient. David Wells, a recognized Full Stack developer of Serverless Architectures puts it succinctly, “Serverless tech pushes us into a world where we try to streamline everything we are doing, where we want our program to be as efficient as possible.”

It's well known that new technologies are never perfect out of the gate. You can't expect to invent a whole new paradigm for software development that is free from technical issues. It's the nature of the beast. Even with its ever-growing popularity, serverless is no exception, and it has its own hurdles to overcome.

The top four issues in serverless today require creative solutions to solve:

**Performance**
- Cold starts create latency (especially in applications where a chain of functions are invoked one after another).

**Security**
- Existing serverless platforms do not secure the application level.

**Monitoring**
- Inability to monitor applications and functions in real time, to troubleshoot problems and to track cost.

**Debugging**
- Log-based performance metrics provided by vendors are insufficient for proper code debugging.
Performance / Cold Start

To maximize performance and avoid long cold-starts as much as possible, FaaS providers (such as AWS Lambda) keep containers alive for up to 55 minutes after a function has completed running, just in case the function will be needed again. That’s great for reducing latency – warm-starts are, by definition, faster than cold-starts – but it can have troubling effects on your serverless functions.

For a start, container reuse could result in functions running out of memory because of problems in the cleanup process. This is never good. Furthermore, keeping the containers up and running encourages attacker persistence because it enables attackers to hang around for longer periods of time. This cancels out one of the main advantages of a serverless system. And when containers are reused, attackers can spread themselves across more of the system. For example, if an attacker injects poisonous code into a function, the poison can spread through the architecture if the containers are reused.

By design, when services are not often used, they are unloaded from memory. So what are your choices? You can either go with low-performance but safer cold starts, or you can keep the code in a ready “warm” state (with high financial and security costs).

You can either go with low-performance but safer cold starts, or you can keep the code in a ready “warm” state (with high financial and security costs).
Serverless Security

Because serverless is new, and because security is a hot topic, one can be forgiven for assuming that serverless solutions have sufficient security defenses baked in from the get-go. After all, take a look at who the providers are: established, trusted, solid, clever organizations, such as Amazon and Microsoft.

The assumption of built-in serverless security, however, is wrong and many organizations are unaware that existing platforms do not fully secure serverless. For example, cloud providers like AWS Lambda, Azure Functions, and GCF don't natively secure the application level. This means that when companies go all-in on serverless, their topmost layer - the most valuable part of their apps - is left exposed. Yet serverless platforms are being adopted at an increasing rate with security being given little thought. Additionally, as the organizational power shifts towards developers, security is increasingly compromised in favor of development velocity and simplicity.

The assumption of built-in serverless security is wrong and many organizations are unaware that existing platforms do not fully secure serverless.

But even if the developers themselves wanted to install conventional application security tools and other measures to address these issues, they can't. One of the foundational aspects of serverless is that the infrastructure is managed by the serverless provider so that the developers don't have to worry about it and can get on with their core tasks. The result of this is that there is no access to the underlying infrastructure or servers, so there is no way to add your own security layer to your serverless platform, even if you wanted to.

This misplaced organizational trust in serverless providers is possibly due to the lure of the advantages that serverless offers. Consumers of serverless platforms are increasingly recognizing the systematic flaws in the architectures of the serverless systems that they are buying into (like container reuse and storage of credentials in the functions' containers, such as IAM roles), and are actively searching for viable and safe alternatives to current systems.
Monitoring

Serverless platforms mean that you don’t have to set up, run, provision or maintain servers. The whole point of serverless is for you to concentrate on code and let the FaaS provider do the rest. The improvements in development times, though, have a downside. Without control over the infrastructure, you give up the ability to closely monitor your environment and the functions within them, to troubleshoot problems, and to track costs in real-time or otherwise.

Effective monitoring provides the most relevant information at the right time. It also reduces the time and energy required to properly map systems and understand how well they are working.

Succinctly described, “With serverless computing, when a problem arises, developers can lack visibility into every layer of the system. This can pose a major issue for troubleshooting and can impact customer experience. Often the only way to effectively discover root causes in distributed cloud-based systems is to have access to the right data at the right time.”

When the environment on which applications are actually executed is locked down, and it is difficult to install agents and daemons for monitoring purposes, it's like driving blind – you don’t know where you are, where you’re going, or how well you are doing until something goes terribly wrong.

This can also become a security issue because an undiscovered or open error is just another weak point where an attacker can take advantage, so it’s critical to gain real-time visibility to functions and activities.

When the environment on which applications are actually executed is locked down, it’s like driving blind.
Debugging

Normally we write our code and QA it locally. But to really know how well the code performs, you need to run it on the system and in the environment that is going to eventually host it.

That’s the problem with testing locally and then sending the production version to the cloud. The challenge here is that even though serverless vendors offer log-based performance metrics (like AWS CloudWatch), it is very limited in what it can do in terms of debugging, and there’s no way to attach debuggers to AWS Lambda. Considering the cost of logging, log-storage, and log-processing, it can also turn out to be quite expensive.

Even though serverless vendors offer log-based performance metrics, it is very limited in what it can do in terms of debugging.

It is possible to test and debug Lambdas locally, but with great difficulty. In the end, “the answer to debugging in production is to have a well-thought-out logging strategy, use it consistently, and fetch your logs from CloudWatch when you need to figure out what’s going on.” Therefore you can test locally, but your only hope of knowing how your Lambdas perform in a “real” environment is to examine the CloudWatch logs. We’re back to where we started.

So your debugging options are limited. Either you develop and QA locally, and then cross your fingers that the code will perform in the production environment as intended, or you can use CloudWatch logs that provide some benefit, but that is missing a whole lot of really useful information.
Conclusion

Serverless architectures like AWS Lambda are attractive to application developers. They provide a vehicle for speeding up the development cycle and enable developers to concentrate on what matters to them most: creating great apps. But the inherent problems of security, monitoring, debugging, and performance are constant pain points that cannot – and should not – be ignored.

About Nuweba

Nuweba enables organizations to reap the benefits of serverless, without compromising on security or efficiency. And serverless performance is enhanced with blazingly fast cold-start times.

Nuweba secures your serverless functions by employing advanced detection and prevention mechanisms, including operating system function hardening, controlling and inspecting communications and networking, and heuristic behavioral detection.

With real-time detection and mitigation of application-level attacks, along with deep operational insights into serverless functions, Nuweba is a zero-overhead solution, requiring no changes to your functions or code.

Nuweba secures serverless.