Serverless Architecture: Platforms, Benefits & Best Practices

Learn what is serverless architecture, how serverless platforms work, pros and cons of serverless, and best practices for improving serverless security

What is Serverless Architecture?

Serverless is a development model that lets developers run code in a scalable manner without having to manage servers.

Serverless is a commonly used component in a microservices architecture, which decomposes applications into small, independent units, each of which does one thing well. Deploying and managing microservices is very convenient in a serverless model.

Unlike an infrastructure as a service (IaaS) model, in which users receive virtual machines (compute instances) and pay for them by the second or hour, in a serverless model users are billed according to actual compute time—the amount of time a serverless function runs.

Serverless has tremendous potential to improve developer productivity and get applications to market faster. At the same time, it raises serious challenges like the difficulty of monitoring and troubleshooting serverless applications, and serverless security issues like the need to scan short-lived functions for vulnerabilities, and prevent code injection.

In this article, you will learn:

Serverless Platforms

All major cloud providers, including AWS, Azure and Google Cloud, provide serverless platforms, with the most popular being Amazon Lambda. The cloud provider takes over tasks like provisioning, managing and scaling servers. Users only need to provide code, in any programming language, and run it, typically in response to event triggers.

AWS Lambda

AWS Lambda is a serverless platform that enables developers to run code in any programming language. It does this using anonymous functions called Lambda functions.

Key features include:

- **Supports any deployment artifact**—lets you run code in JavaScript, Python, Java, or any other programming language, or alternatively, use Lambda functions to run libraries or executable files, availability zone.

- **Event driven**—lets you trigger Lambda function using events from a variety of Amazon services, including changes to files in S3, DynamoDB, CodeCommit (triggering a function as a result of changes to a build), CloudWatch (triggering functions when any alert is raised in your AWS environment), streaming data from SNS or Kinesis, etc.

Azure Functions

Azure Functions is Microsoft's serverless platform, which lets you run applications in a serverless model in Azure, fully integrating with other Azure services or compatible Microsoft services in your on-premise data center. It provides the same level of security and compliance as the rest of the Azure cloud.

Key features include:

- **Trigger functions**—use several types of trigger functions to invoke a function, including messaging queues, timers, and HTTP requests.
- **Browser-based user interface**—lets you write code in a web-based IDE or in popular development tools.
- **Supports continuous deployment and integration**—supports CI/CD through integration with GitHub, Visual Studio Team Services, tools like Eclipse and Xcode.

Google Functions

Google Cloud Functions is a serverless execution environment, which runs code in a fully controlled environment. Like AWS and Azure, it enables triggering serverless functions by events from other Google services, or external systems.

At the time of this writing, Cloud Functions supports only Node.js, Python, Go, Java and .NET.

Key features include:

- **Fine-grained pay per use**—pay per function execution time, rounded to the nearest 100 milliseconds.
- **Supports open source technologies**—lets you run functions using open source serverless frameworks based on Knative.
- **End-to-end visibility**—comes with monitoring tools that provide full observability of serverless applications, and support for debugging on local workstations.

Other Serverless Platforms

There are many more cloud services that involve a serverless delivery model. A few examples are:

- **Amazon Fargate**—delivers Docker containers in a serverless model, with servers completely abstracted from the user
- **Amazon Aurora Serverless**—an elastically scalable version of the Aurora database service, which transparently manages database instances
- **Azure CosmosDB**—a NoSQL database service based on a serverless infrastructure

Serverless Architecture Pros and Cons

Let's review some of the key advantages and disadvantages of a serverless computing model.

Advantages of a Serverless Architecture

Less Maintenance

In addition, serverless providers have a backend infrastructure that can automatically scale horizontally and vertically to meet demand, with no need for manual configuration, maintenance or monitoring of scaling activity. In an IaaS model, autoscaling capabilities are available but they require expertise to set up and require ongoing management.

Lower Costs

Serverless allows organizations to use resources exactly when needed, instead of renting a fixed number of servers for a predefined period of time. In a serverless model, companies pay according to metrics like code execution count, memory used, and execution time, and not for idle time when code is not running. This can save cost substantially compared to an IaaS model.

Disadvantages of Serverless Architecture

Vendor Lock-In

With serverless, organizations lose control over their infrastructure. Entire applications may be written against a serverless framework with proprietary APIs, which makes it very difficult to migrate to another provider. Google's support for open source frameworks like Knative is a step towards resolving this problem.

Performance

Many teams getting started with serverless ignore application performance metrics, because they don't have servers to manage. However, serverless applications still experience latency issues, as a result of connectivity problems, code inefficiencies, or problems with systems that generate events upstream.

It's important to remember there is a server behind a serverless system, but it is not visible to the organization. Organizations using serverless applications need to monitor and control traffic so that users do not experience performance issues, and they need new ways to diagnose performance issues without having access to the physical server.

Serverless Architecture Examples and Use Cases

Here are a few examples of how serverless powers real life applications on AWS.

Analytics Stream Processing

On AWS, you can use Amazon Athena to analyze S3 data, and generate events based on the results of your queries. Athena is a serverless solution that allows you to query large amounts of data and get results in seconds, and you only pay for the scanned data. You can integrate Athen with a BI service to visualize query results.

To extend this environment, use Kinesis Data Streams to collect analytic events and process them in real time using Lambda functions, and use Kinesis Data Firehose to collect the events and place them in your data lake. Whenever new events are loaded to S3, this can trigger additional Lambda functions for further processing.

Web Applications

You can build a single page application (SPA) on the front end and implement a backend API using Amazon API Gateway, Lambda, and a serverless-compatible database like DynamoDB. You can also use Lambda to integrate the AppSync API with other data sources (e.g. Aurora).

All of the above services offer built-in scalability and flexibility. For example, by default, Lambda functions are deployed across multiple Availability Zones (AZs) and scale automatically based on traffic.

IoT

Serverless technologies allow small teams to focus on delivering functionality instead of wasting time on infrastructure. When new devices are developed or new types of data are introduced to the system, it is easy to add new types of processing and analytics for them, using serverless functions that act on triggers from IoT sensors.

Serverless Security Best Practices

Serverless security is a major concern for organizations starting to roll serverless into production. Traditional security tools are not effective in a serverless environment, because they are focused on protecting servers or static applications. Here are a few quick best practices that can help you improve serverless security.

Adopt the Principle of Least Privilege

Because serverless functions represent small pieces of functionality, you can minimize the set of permissions granted to each function. Only provide permission to a function if it is absolutely necessary for it to do its job. This lets you significantly reduce damage during a successful attack, and minimize the attack surface.

For example, most functions may not require access to a database or external servers, and so these permissions should be blocked. These are actions typically taken by an attacker or a malicious user after a successful exploit.

Scan Functions for Vulnerabilities

Scan function code to identify malware, and look for security vulnerabilities from sources like the Common Vulnerabilities and Exposures (CVE) database, and security announcements by tools and open source products you use. Monitor keys and secrets stored by functions, and ensure they are encrypted to prevent accidental disclosure.

All the above is difficult to achieve in serverless applications, and requires dedicated security tools designed for a cloud native environment.

Use Runtime Security

The execution time of serverless functions is usually very short, measured in minutes or seconds. This is good for security, because many attack patterns rely on persistency of the target system.

However, there are several new attack options that allow attackers to use serverless functions for illicit activity, such as cryptocurrency mining or spamming, or use the function as a springboard to access other resources in your cloud account.

These attacks typically use code injection techniques, or serialization attacks, which are unavoidable during the development and deployment stages. To prevent such attacks, use tools that can track functional behavior and block threats in real time.

Serverless Security with Aqua

Aqua provides the industry's most complete security platform for cloud native applications, from development to production. The platform secures applications based on VMs, containers, and serverless, across the entire development to production lifecycle. Using dedicated controls for serverless that are optimized for performance and scale, Aqua allows organizations to gain visibility into serverless risk, mitigate it and protect serverless applications against attacks.

If you are running cloud native workloads on serverless, in AWS or Azure, Aqua can help you with:

- **Scanning functions for vulnerabilities, malware, and secrets**—ensure that functions are scanned for known vulnerabilities in OS and programming language packages, detect malware and embedded "secrets" such as AWS keys and tokens.

- **“Shifting left” into the CI pipeline**—integrate into the developers’ CI pipeline to make it easy to detect issues in functions as they are built, reducing time to detect and fix issues.
- **Preventing non-compliant functions from being used**—use Aqua’s advanced assurance policies to create custom rules for tagging functions as non-compliant (e.g., based on vulnerability severity, secrets, permissions) and blocking them from running.
- **Protection functions against code injection and attack**—using Aqua’s advanced Drift Prevention, ensure that no new code can be added or written into a function while it is running
- **Identifying Indicators-of-Compromise (IoCs) in functions**—using Aqua’s unique honeypots, embed honeypot AWS credentials as traps in your functions, and automatically monitor their use, getting clear indication of malicious attempts to compromise the functions.

Learn more about Aqua for serverless security ›

Aqua Blog All about cloud native and security