sogeti

Part of Capgemini

Serverless Architecture

Evolving as Future Business Solutions

July - 2019

Table of Contents

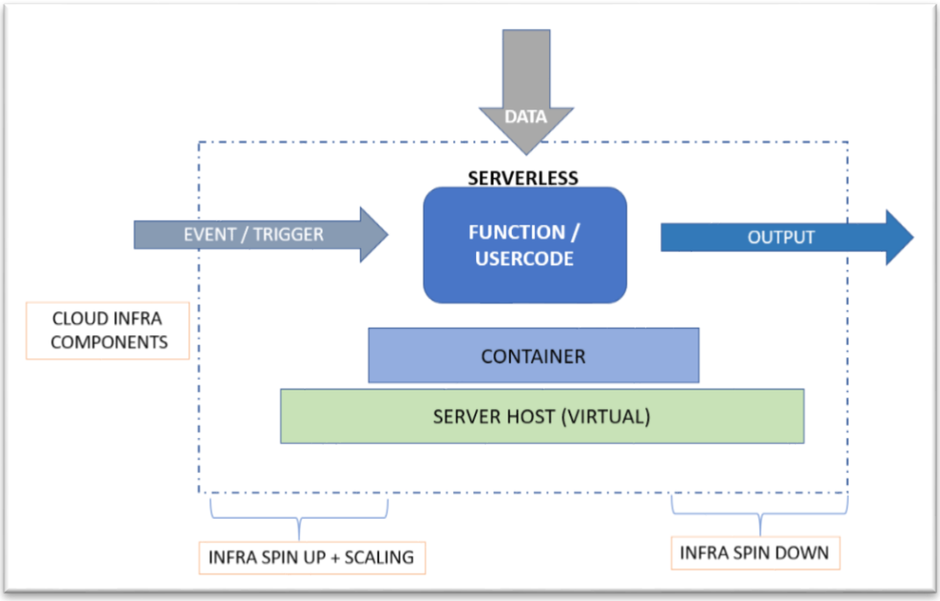
Introduction

Serverless architecture is a concept, wherein developers earlier developed functionalities which were dependent on serverside computing can now be developed and hosted on the client side. This eliminates the dependency on serverside components for development.

The serverless framework helps to develop and deploy serverless applications by using Cloud Platform services. Every serverless service instance is an independent unit of deployment, for eg. - microservice.
It's a code, deployed in the cloud, that is written to perform a single job.

Serverless computing, in a nutshell, allows developers to define a code function that a cloud service executes as a result of an event. The code could be Java, Python, C# or a variety of other languages, along with appropriate dependencies, which are grouped (perhaps in a .ZIP file) and uploaded to a storage service, ready to be executed when needed. Some "functions" can also be coded directly into the service via a GUI.

The Cloud vendor provides the Function as a Service model that enables the platform to provide the underlying infrastructure (typically a container on an already-provisioned VM or service) and executes the code.



An event – for example, a file is placed into a storage service – triggers the serverless function code.

The cloud platform hydrates the needed infra, launches the code, monitors the activity, scaling as required and finally dehydrates the infra on process completion.

Business case

Serverless is an ideal back-end technology. Obviously, serverless has strengths in analytics, artificial intelligence (AI) and IoT, but there's nothing to stop it also being used for many of the compute functions we commonly associate with VMs today.

- **Application backends (Web & Mobile) -** Display a webpage when triggered by an HTTP event, to make API calls to clouds as a result of time, poll or user event.
- **Real-time Processing -** Includes performing analysis on datasets (either on a regular schedule or when a dataset is uploaded), cleaning up a dataset such as a log or IoT data prior to processing (particularly when that data has to be transmitted)
- **Workflows and Automation -** Automating backups and other daily tasks and processing data including scheduled tasks.
- **Extending & Integrating Systems -** To interact with any other number of services and systems in the enterprise or across enterprises

Pros

Benefits around the Serverless model are largely around the following key areas –

- No server management is necessary as developers never have to deal with the servers. They are managed by the vendor. This can reduce the investment necessary in **DevOps**, which lowers expenses, and it also frees up developers to create and expand their applications without being constrained by server capacity.
- Developers are only charged for the server space they use, reducing cost as in a **'pay-as-you-go'** phone plan, developers are only charged for what they use. Code only runs when backend functions are needed by the serverless application, and the code automatically scales up as needed. Provisioning is dynamic, usually precise, and real-time.
- Serverless architectures are inherently scalable since applications built with a serverless infrastructure will scale automatically as the user base grows or usage increases. If a function needs to be run in multiple instances, the vendor's servers will start-up, run, and end them as they are needed, and these are **faster spinning** up as they often using containers-based models.
- This also fits well with microservices architecture development paradigm as the Functions model can be used to leverage Request based API model with clear **separation** of concerns and business actions.
- The complexity of software decreases as the applications and solution will necessarily have to be broken up into well-defined bits that can operate **independently** while integrating into the larger whole.
- Quick deployments and updates are possible as there is no need to upload code to servers or do any backend configuration to release a working version of applications. Developers can upload code all at once or one function at a time since the application is not a single **monolithic** stack but rather a collection of functions and each provisioned by the vendor.
- The code can run closer to the end-user, decreasing **latency** because the application is not hosted on a fixed predetermined origin server and that means its code can be run from anywhere. It is therefore possible, depending on the vendor used, to run application functions on servers that are close to the end-user reducing latency.

Cons

- The serverless computing model cannot be stated as the perfect approach due to performance issues. The model itself implies that there will be higher latency in how the compute resources respond to the applications' requirements. In such cases, it would be better to use **allocated virtual servers** if performance is the primary requirement.
- The debugging and monitoring of serverless computing are tricky as well. As it is typically utilizing a **single server resource**, the debugging and monitoring activities become extremely difficult.

Regarding this, the tools that manage debugging and monitoring in the serverless environments will arrive eventually.

- Around Security practices, when vendors run the entire backend, it may not be possible to fully vet their security, which can especially be a problem for applications that handle personal or sensitive data. Because companies are not assigned their own discrete physical servers, serverless providers will often be running code from several of their customers on a single server at any given time. This issue of sharing machinery with other parties is known as **'multitenancy'** and can affect application performance and, if the multi-tenant servers are not configured properly, could result in data exposure.
- As Serverless architectures are not built for long-running processes, this limits the kinds of applications that can cost-effectively run in a serverless architecture. Therefore, as serverless providers charge for the amount of time code is running, it may cost more to run an application with **long-running processes** in a serverless infrastructure compared to a traditional one.
- Performance may be affected because it's not constantly running and the serverless code may need to 'boot up' when it is used. This **start up time** may degrade performance. However, if a piece of code is used regularly, the serverless provider will keep it ready to be activated – a request for this ready-to-go code is called a 'warm start.' A request for a code that hasn't been used in a while is called a 'cold start.'
- Vendor lock-in is a risk since allowing a vendor to provide all backend services for an application inevitably **increases reliance** on that vendor. Setting up a serverless architecture with one vendor can make it difficult to switch vendors if necessary, especially since each vendor offers slightly different features and workflows.

Cloud Providers

AMAZON WEB SERVICES	AWS Lambda	Market-maker AWS had a year-and-a-half head start on everyone else. Led by Node.js, AWS has since added Java and Python to better address enterprise developers and DevOps engineers, respectively.
MICROSOFT	Azure Functions	Azure Functions is built atop the Azure WebJobs SDK. The service supports C# and JavaScript/Node.js, Python, F#, PowerShell, PHP, Bash and more. Today it also supports a number of integrations, plus timers and arbitrary applications via webhooks.
GOOGLE	Google Cloud Functions	GCF supports Node.js at this point. It can be triggered by any Google service that supports cloud pub/sub (Google emphasizes logging and email), cloud storage, arbitrary webhooks and direct triggers.
IBM	IBM OpenWhisk	This served as a clear contrast to Lambda because it was runnable not only in the public cloud but also in any Bluemix environment. While OpenWhisk initially depended on IBM's Cloudant offering, CouchDB was quickly added as an option, which enabled running the FaaS without paying IBM.
OpenFaaS	OPENFAAS	OpenFaaS (Functions as a Service) is a framework for building serverless functions with Docker and Kubernetes. OpenFaaS is free to use and completely open source under the MIT license.

Technical Evaluation

Ecosystem

The AWS Serverless Application Repository makes it easy for developers and enterprises to quickly find, deploy, and publish serverless applications in the AWS Cloud.
You can also easily publish applications, sharing them publicly with the community at large, or privately within your team or across your organization.

Azure has ready-made templates that provide quick starters for creating a new Serverless function app on the Cloud

Design Considerations

A few considerations to keep in mind while designing Serverless/Function based systems are -

- **Develop single-purpose stateless functions**: Since functions are stateless and persist for a limited duration only, it is recommended to write single-purpose codes for function. Though statelessness could be viewed as a limitation, it provides infinite scalability to a platform to handle an increasing number of requests, which otherwise would not have been possible. Testing is also limited in the context of Serverless as the platform only provides limited accessibility.
- **Develop for a Microservices based model for larger systems**: Plan and design larger application from the ground up to leverage a decoupled, microservices-based pattern to develop, deploy, scale and test independently.
- **Design push-based**, event-driven patterns for integration systems: Designing push-based and event-driven architecture patterns where a chain of events propagate without any user input imparts scalability to an architecture.
- **Focus on limited Function run times**: Understand the vendor provided limits and recommendations to plan and design the Functions and limit execution times. This limiting of the execution time of a function also has a direct impact on cost.
- **Thicker and/or powerful UI/Presentation layers** around both Microservices and Integration models: Bring execution of more complex functionality at the front-end especially through rich client-side application framework helps reduce cost by minimizing function calls and execution times. Completely decoupling back-end logic from the front-end. This also allows more services to be accessed from front-end resulting in better application performance and richer user experience.
- **Identify performance bottlenecks**: On-going measurement of performance bottlenecks in terms of identifying which functions are slowing down a particular service is critical to ensure optimal customer experience.
- **DevOps based Agile Development and Faster Release**: Single purpose codes, Functions, are easier to test, deploy and release thus improving enterprise agility.
- **Incorporate appropriate security mechanism** across the technology stack: Appropriate security mechanisms must be incorporated at the API Gateway layer and also at the FaaS layer. These security mechanisms include features like access controls, authentication, identify and access management, encryption and establishing trust relationship etc.
- **Third party services as needed**: Serverless being an emerging field existing enterprise tools for various services like logging, monitoring etc. may not be compatible. Choosing the right third-party tools for executing the task at hand will be key for enterprises to ensure the benefits of serverless are utilized to the fullest.

Security

Serverless relieves users from managing servers and security updates by shifting this responsibility to the cloud service provider.

User responsibility begins with the code for your application code. Badly written code is not secure. In addition, there is responsibility for data management, data encryption, identity management, authentication/authorization, and configuration of services and role-based access control (RBAC) where the platform supports it.

Monitoring & Logging

AWS Lambda automatically monitors Lambda functions on your behalf, reporting metrics through Amazon CloudWatch. To help you monitor code as it executes, Lambda automatically tracks the request metrics and errors to publish to the associated CloudWatch metrics. You can leverage these metrics to set CloudWatch custom alarms. Users can insert logging statements into the code to help validate the code is working as expected. Lambda automatically integrates with Amazon CloudWatch Logs and pushes all logs from the user code to CloudWatch Logs.

Application Insights (AI) is an application performance management service that helps us in monitoring the performance of an application hosted anywhere. Once you integrate AI into your application, it will start sending telemetry data to your AI account hosted on the cloud.

Economics

The extremely granular pricing model for Serverless/FaaS is likely to provide a strong pull, particularly in comparison with substitutes that offer less granular or even unmetered pricing, compared to the old-fashioned VMs most cloud providers use to deliver compute.

There are two primary cost benefits of serverless technology. First, developers and operations are no longer responsible for infrastructure – developers only need to write the code they want to execute, with no need to spin up servers, install libraries, configure network and security groups or scale code up or down. The benefit of serverless is that users don’t have to spend time procuring and installing any server, physical or virtual.

The second economic benefit comes from increased utilization. With Serverless, the user is only charged for the time they are actively using the platform. With IaaS, a developer needs to have a VM up and running to ensure that code can be executed quickly; thus, there will be times when the VM is idle, and this is sunk cost – wasted expenditure. Even with auto-scaling of servers, there is usually a buffer of over-provisioned resources that exist to provide capacity while additional VMs are spun up. And even as the VMs scale, unutilized capacity continues to ramp up in large steps. This is not an issue with serverless technology.

Price models are charging based on three key metrics, on a monthly basis:

- ➢ The duration for which the code is executing
  In the same way a VM is charged for the minutes or hours it is alive, serverless is charged for the aggregate sum of the time the code is executed for over a month. Each execution duration is rounded up, often to the nearest 100ms. If a code segment of 60ms (rounded up to 100ms) is executed 1,000,000 times during a month, the total duration is 100,000 seconds.

- ➢ The resources assigned to that code during execution
  AWS, Microsoft and IBM associate memory allocations with code execution. AWS and IBM users must define how much memory must be allocated to that code from 128MB to 1.5GB; Microsoft measures the memory consumed and rounds up to the nearest 128MB. Microsoft’s approach is nice because the employee doesn’t have to worry about performance issues if too little memory is assigned or wasted capacity if too much.
  Duration and resource consumption are combined into GB-seconds – essentially how many GBs for how many seconds of RAM are being consumed.
  Google also charges for processor consumption, in units of CPU clock frequency. However, users can’t choose their specific combination of memory and CPU; there are five bundles with fixed size allocations of CPU and memory.

Microsoft has the option of its App Service plan, which allows functions to be executed on dedicated VMs, with scaling performed by adding more VMs. This can be a good choice for those with security concerns or the desire to scale manually to a high level of utilization.

- The number of times the code is executed
  All cloud providers also charge for the number of executions of each code, per request.

It is essentially the same model utilized by VMs, in which size and running time are the basis for cost, with the inclusion of number of times to represent the more variable aspect of serverless. In fact, the conceptual similarity to VM pricing might aid serverless' adoption with enterprises.
Another point to keep in mind is that there are likely to be other costs related to the execution of a code segment, including bandwidth, object storage, support, etc. as an application rarely uses just a single cloud service, and these should be factored into all overall application expenditure analyses.

In the case of the OpenFaaS, the cost is essentially the hardware and software for the infrastructure needed to deploy the framework and there would no monthly or execution based charges.

Conclusion

In my humble opinion the concept of serverless computing via serverless architectures. Architects (Infra, Cloud, Application and Solution) need to evaluate and Design Hybrid solutions basis of future costing factors, Developers can leverage this to deploy an individual "function", action, or piece of business logic which are a quick running process and not add more functions which are expected to grow lengthier.