Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 09, September 2022

A Research Paper on Serverless Computing

Vaishnavi Kulkarni
Technical Support Engineer
Supermoney, Mumbai,India

***Abstract*— Serverless computing is a method of providing backend services on an as-used basis. A serverless provider allows users to write and deploy code without the hassle of worrying about the underlying infrastructure. Keywords— Serverless computing, Cloud computing, front end, back end.**

I. INTRODUCTION

Serverless computing is a method of providing backend services on an as-used basis. A serverless provider allows users to write and deploy code without the hassle of worrying about the underlying infrastructure. A company that gets backend services from a serverless vendor is charged based on their computation and do not have to reserve and pay for a fixed amount of bandwidth or number of servers, as the service is auto-scaling. Note that despite the name serverless, physical servers are still used but developers do not need to be aware of them.

In the early days of the web, anyone who wanted to build a web application had to own the physical hardware required to run a server, which is a cumbersome and expensive undertaking.

Then came cloud computing, where fixed numbers of servers or amounts of server space could be rented remotely. Developers and companies who rent these fixed units of server space generally over-purchase to ensure that a spike in traffic or activity will not exceed their monthly limits and break their applications. This means that much of the server space that gets paid for can go to waste. Cloud vendors have introduced auto-scaling models to address the issue, but even with auto-scaling an unwanted spike in activity, such as a DDoS Attack, could end up being very expensive.

Fig:Cost Benefits of Serverless

Serverless computing allows developers to purchase backend services on a flexible 'pay-as-you-go' basis, meaning that developers only have to pay for the services they use. This is like switching from a cell phone data plan with a monthly fixed limit, to one that only charges for each byte of data that actually gets used.

The term 'serverless' is somewhat misleading, as there are still servers providing these backend services, but all of the server space and infrastructure concerns are handled by the vendor. Serverless means that the developers can do their work without having to worry about servers at all.

The term 'serverless' is somewhat misleading, as there are still servers providing these backend services, but all of the server space and infrastructure concerns are handled by the vendor. Serverless means that the developers can do their work without having to worry about servers at all.

II. ARCHITECTURE OF SERVERLESS COMPUTING

Serverless architecture is largely based on a Functions as a Service (FaaS) model that allows cloud platforms to execute code without the need for fully provisioned infrastructure instances. FaaS, also known as Compute as a Service (CaaS), are stateless, server-side functions that are event-driven, scalable, and fully managed by cloud providers.

DevOps teams write code that focuses on business logic and then define an event that triggers the function to be executed, such as an HTTP request. The cloud provider then executes the code and sends the results to the web application for users to review.

AWS Lambda, Microsoft Azure Functions, Google Cloud Functions and IBM OpenWhisk are all well-known examples of serverless services offered by the cloud providers.

The convenience and cost-saving benefits associated with on-demand auto-scaling resources, and only paying for services as they're needed, makes serverless frameworks an appealing option for DevOps teams and business stakeholders alike.

- Examples of serverless

The growing popularity of cloud computing and microservices combined with the demand for greater innovation and agility without increasing costs has contributed significantly to the prevalence of serverless applications. Notable use cases include:

- Slack:

Serverless is ideal for independent task based applications such as chatbots and can save on operational costs since billing is based on the actual number of requests. Slack, a popular, cloud-based business communication platform, uses a serverless application called marbot to send notifications from Amazon Web Services (AWS) to DevOps teams through Slack.

- HomeAway:

Reducing development time and server costs while simplifying the build process are goals that universally appeal to business teams and IT teams. HomeAway relied on Google Cloud Functions to develop an app that allowed users to search and comment on the recommendations of travelers in real time, even in areas without an internet connection. The cloud services available through Cloud Firestore and Cloud Functions made it possible to set up the infrastructure within

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 09, September 2022

minutes and deploy the app within six weeks with just one full-time developer.

- GreenQ:

Garbage pick-up and disposal is an industry that may not seem to require innovative technology, but GreenQ took a sophisticated approach to streamlining and improving waste management by using IBM OpenWhisk to create an IoT platform that uses hardware installed on garbage trucks to collect key metrics such as pickup time, location, and load weight. The auto-scaling available through serverless was particularly valuable due to the fluctuation of infrastructure demands based on the number of customers and trucks at any given time.

- Coca-Cola:

Soft drink giant Coca-Cola has enthusiastically embraced serverless after its implementation in vending machines resulted in significant savings. Whenever a beverage is purchased, the payment gateway makes a call to the AWS API Gateway and triggers an AWS Lambda function to complete the transaction. Since vending machines must communicate with headquarters for inventory and marketing purposes, the ability to pay per request rather than operating at full capacity had a substantial impact on reducing costs.

III. FRONTEND VS BACKEND

Application development is generally split into two realms: the frontend and the backend. The frontend is the part of the application that users see and interact with, such as the visual layout. The backend is the part that the user doesn't see; this includes the server where the application's files live and the database where user data and business logic is persisted.

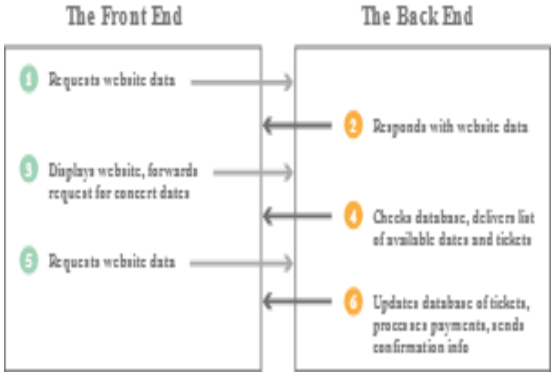

Fig :Frontend vs Backend

For example, let's imagine a website that sells concert tickets. When a user types a website address into the browser window, the browser sends a request to the backend server, which responds with the website data. The user will then see the frontend of the website, which can include content such as text, images, and form fields for the user to fill out. The user can then interact with one of the form fields on the frontend to search for their favorite musical act. When the user clicks on 'submit', this will trigger another request to the backend. The backend code checks its database to see if a performer with this name exists, and if so, when they will be playing next, and how many tickets are available. The backend will then pass that data back to the frontend, and the frontend will display the results in a way that makes sense to the user. Similarly, when the user creates an account and enters financial information to buy the tickets, another back-and-forth communication between the frontend and backend will occur.

IV. ADVANTAGE AND DISADVANTAGE OF SERVERLESS COMPUTING

Advantage

- Lower costs - Serverless computing is generally very cost-effective, as traditional cloud providers of backend services (server allocation) often result in the user paying for unused space or idle CPU time.
- Simplified scalability - Developers using serverless architecture don't have to worry about policies to scale up their code. The serverless vendor handles all of the scaling on demand.
- Simplified backend code - With FaaS, developers can create simple functions that independently perform a single purpose, like making an API call.
- Quicker turnaround - Serverless architecture can significantly cut time to market. Instead of needing a complicated deploy process to roll out bug fixes and new features, developers can add and modify code on a piecemeal basis.

Disadvantage

- Testing and debugging become more challenging

It is difficult to replicate the serverless environment in order to see how code will actually perform once deployed. Debugging is more complicated because developers do not have visibility into backend processes, and because the application is broken up into separate, smaller functions.

- Serverless computing introduces new security concerns

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' – think of several companies trying to lease and work in a single office at the same time. Multitenancy can affect application performance and, if the multi-tenant servers are not configured properly, could result in data exposure. Multitenancy has little to no impact for networks that sandbox functions correctly and have powerful enough infrastructure.

- Serverless architectures are not built for long-running processes

This limits the kinds of applications that can cost-effectively run in a serverless architecture. Because 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

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 09, September 2022

Because it's not constantly running, serverless code may need to 'boot up' when it is used. This startup 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 code that hasn't been used in a while is called a 'cold start.'

- Vendor lock-in is a risk

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

V. How does serverless compare to other cloud backend models?

A couple of technologies that are often conflated with serverless computing are Backend-as-a-Service and Platform-as-a-Service. Although they share similarities, these models do not necessarily meet the requirements of serverless.

Backend-as-a-service (BaaS) is a service model where a cloud provider offers backend services such as data storage, so that developers can focus on writing front-end code. But while serverless applications are event-driven and run on the edge, BaaS applications may not meet either of these requirements.

Platform-as-a-service (PaaS) is a model where developers essentially rent all the necessary tools to develop and deploy applications from a cloud provider, including things like operating systems and middleware. However, PaaS applications are not as easily scalable as serverless applications. PaaS also don't necessarily run on the edge and often have a noticeable startup delay that isn't present in serverless applications

Infrastructure-as-a-service (IaaS) is a catchall term for cloud vendors hosting infrastructure on behalf of their customers. IaaS providers may offer serverless functionality, but the terms are not synonymous.

CONCLUSION

Serverless computing continues to evolve as serverless providers come up with solutions to overcome some of its drawbacks. One of these drawbacks is cold starts. Typically when a particular serverless function has not been called in a while, the provider shuts down the function to save energy and avoid over-provisioning. The next time a user runs an application that calls that function, the serverless provider will have to spin it up fresh and start hosting that function again. This startup time adds significant latency, which is known as a 'cold start'.

Once the function is up and running it will be served much more rapidly on subsequent requests (warm starts), but if the function is not requested again for a while, the function will once again go dormant. This means the next user to request that function will experience a cold start. Up until fairly recently, cold starts were considered a necessary trade-off of using serverless functions.

REFERENCES

[1] https://martinfowler.com/articles/serverless.html
[2] http://lukeangel.co/cross-platform/docker-servless-faas-functions-as-a-service/
[3] https://hackernoon.com/what-is-serverless-architecture-what-are-its-pros-and-cons-cc4b804022e9
[4] https://serverless.com/blog/2018-serverless-community-survey-huge-growth-usage/
[5] https://serverless.com/framework/docs/providers/