

## **Advanced System Analysis and Design**

**Hot Topic Report on**

### **Serverless Computing Function as a Service(FaaS)**

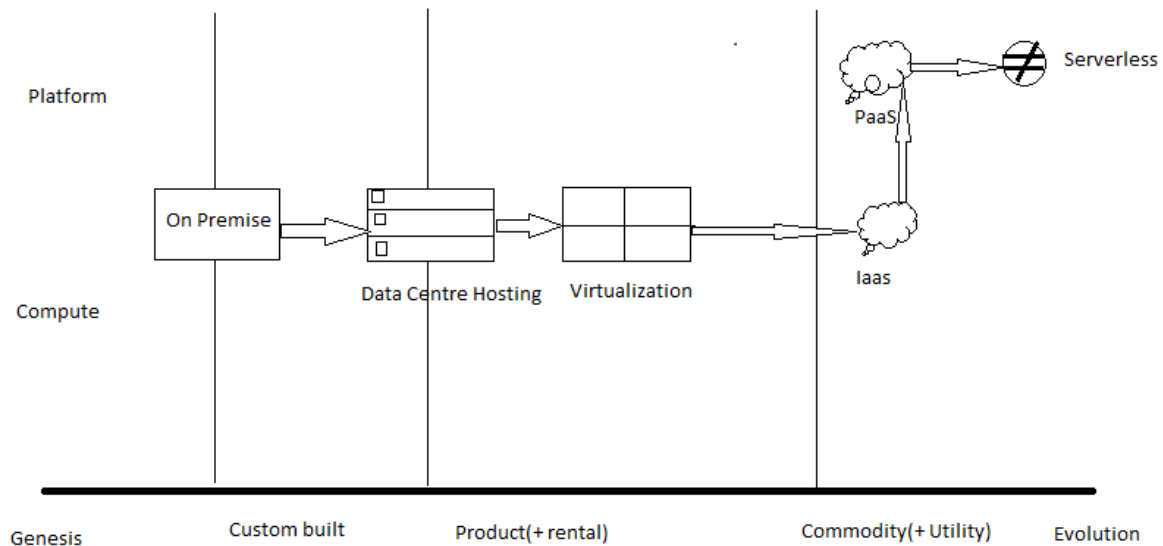


## Contents

Hot Topic Report on .....	1
Abstract.....	3
Overview of Serverless Computing:.....	4
What is driving the trend of Serverless Computing? .....	5
Advantages of Serverless computing for business: .....	6
Introduction to AWS Lambda: .....	7
Introduction to AWS Lambda functions: .....	7
Key Features of AWS Lambda: .....	8
Completely Automated Administration .....	9
Built-in Fault Tolerance .....	9
Automatic Scaling.....	9
Run Code in Response to Amazon CloudFront Requests.....	10
Integrated Security Model.....	10
Bring Your Own Code .....	11
Pay Per Use .....	11
Flexible Resource Model .....	11
Different Use Cases for Serverless Computing or FaaS (Function as a Service): .....	12
Software development challenges using FaaS:.....	16
Conclusion.....	19
References .....	20

## Abstract












The following figure shows the evolution of cloud:



Serverless computing enables you to develop and run applications and administrations without contemplating servers. Serverless applications don't expect you to arrange, scale, and deal with any servers. You can assemble them for practically any kind of utilization or backend operations, and everything required to run and scale your application with high accessibility. Here in the case for FaaS implementation, instead of having to run a server continuously we can implement functions in FaaS that gets triggered only through HTTP requests via API Gateway. Building serverless applications imply that designers can concentrate on their product as opposed to stressing over working servers or runtimes, either in the cloud or on-shore. This decreased overhead gives designers a chance to recover time and vitality that can be utilized judiciously.

AWS Lambda enables you to run code without provisioning or overseeing servers. You pay just for the process time you expend - there is no charge when your code isn't running. With Lambda, you can run code for all intents and purposes any kind of utilization or backend benefit - all with zero organization. Simply transfer your code and Lambda deals with everything required to run and scale your code with high accessibility. You can set up your code to consequently trigger from different AWS administrations or call it specifically from any web or versatile application.

## Overview of Serverless Computing:

FaaS	BaaS	B2D SaaS
      	    	        

The above table shows the major providers in FaaS (Function as a Service), BaaS (Backend as a Service) and B2D SaaS (Business to Developer Software as a Service).

Also referred to as FaaS, the expression "serverless" shows up in an article by Ken Fromm , written in 2012. In any case, Zimki opened the entryway for FaaS by propelling the main Platform as a Service (PaaS). Along these lines, as distributed computing was commanding the innovative world – fuelled by the expanding number of web associated gadgets (IoT) – Amazon Web Services presented Lambda (PaaS) in 2014.

To be sure, "serverless" still includes utilizing servers for all the standard capacities, for example, sending programming, putting away information, et

cetera. In any case, by contracting with a FaaS supplier, server securing, administration, programming refresh, and repair are not anymore your duty (however there are circumstances where a few applications give restricted server-side co-operations with your engineers). Appropriately, both conventional and virtual servers are getting to be noticeably undetectable inside the business world.

Organizations, for example, IBM, Amazon Web Services, Microsoft, and Google give the framework and upkeep exercises ordinarily connected with confined, committed servers. Subsequently, your opportunity and assets are liberated from the difficulties of server design or concerns with respect to your center working framework on the backend. Your concentration would now be able to move to settling the torment directs one of a kind toward your industry, and further situating yourself as a market pioneer.

## **What is driving the trend of Serverless Computing?**

Computer algorithms are progressively reflecting the learning highlights of human knowledge. All things considered, manmade brainpower (AI) is rapidly developing in size and extension. This won't just outcome in bigger measures of information, however, purchaser confronting applications should line up with growing abilities of the gadgets which send AI. Subsequently, it will be inconceivable for designers to keep pace if used on the backend is postponed for any reason.

Designers will, in this way, need the capacity to react to quick changes in the information signals entering each of your information channels. In synopsis, serverless registering enables engineers to make, run, and oversee discrete, directed application capacities without being overloaded by extra weights on the backend. Likewise, on the off chance that you have to scale a specific API endpoint, serverless figuring offices this with laser-like core interest.

Such is the reason that serverless figuring is ceaselessly recorded as one of the best innovation patterns and considering current circumstances.

## **Advantages of Serverless computing for business:**

### **Efficiency:**

- Basically, the DevOps group is centered around composing amazing code as opposed to being worried about the machines running the code.
- On the FaaS framework side, the code is conveyed through occasion activating.
- Execution issues that depend on the danger of under-provisioning (and scrambling to expand server limit) are evaded.
- Since the speedier a code is executed, the less expensive the cost, this can help gauge the productivity of capacity plan. In synopsis, the more productive a designer's code, the less cost is acquired through utilizing the FaaS supplier's framework.

### **Cost:**

- As beforehand talked about, the buying or leasing of servers is costly. At that point include the cost of utilizing people to keep up the equipment and programming framework. With serverless registering, you require not acquire and administer to in-house servers.
- Scaling is less expensive given the focused-on usefulness. FaaS suppliers, for example, Amazon Web Services and IBM auto-scale the assets for you.
- At the point when not being used, the server cost to you is zero. This is not the same as conventional servers that are depleting your pockets notwithstanding when sitting out of gear. In FaaS, you are charged just for the time your code is sent.
- No compelling reason to stress over finished provisioning as it is the supplier's duty to arrangement limit.

## **Introduction to AWS Lambda:**

AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you. You can utilize AWS Lambda to expand different AWS administrations with custom rationale, or make your own back-end benefits that work at AWS scale, execution, and security. AWS Lambda can naturally run code in event of numerous occasions, for example, HTTP asks for through Amazon API Gateway, changes to objects in Amazon S3 cans, table updates in Amazon DynamoDB, and state advances in AWS Step Functions.

Lambda runs your code on high-availability compute infrastructure and performs all the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code and security patch deployment, and code monitoring and logging. All you need to do is supply the code.

## **Introduction to AWS Lambda functions:**

The code you keep running on AWS Lambda is known as a "Lambda function." After you make your Lambda function it is constantly prepared to keep running when it is activated, like an equation in a spreadsheet. Each function consists of your code and adds some related configurational setup data, including the function name and asset prerequisites. Lambda functions are "stateless," with no liking to the infrastructure, so Lambda can quickly dispatch the same number of duplicates of the function as expected to scale to the rate of incoming events

After you transfer your code to AWS Lambda, you can connect your functions with AWS assets (e.g. a specific Amazon S3 can, Amazon DynamoDB table, Amazon Kinesis stream, or Amazon SNS notice). At that point, when the resource changes, Lambda will execute your function and deal with the figure resources as required to stay aware of approaching requests.

## Key Features of AWS Lambda:

### Extend Other AWS Services with Custom Logic



AWS Lambda enables you to add custom logic to AWS assets, for example, Amazon S3 containers and Amazon DynamoDB tables, making it simple to apply compute to information as it is entering or travels through the cloud.

It is anything but difficult, to begin with AWS Lambda. In the first place, you make your function by transferring your code (or building it ideal in the Lambda console) and picking the memory, timeout period, and AWS Identity and Access Management (IAM) part. At that point, you indicate the AWS asset to trigger the function, either a specific Amazon S3 bucket, Amazon DynamoDB table, or Amazon Kinesis stream. At the point when the resource changes, Lambda will run your function and dispatch and deal with the figure resource as required to stay aware of approaching requests.

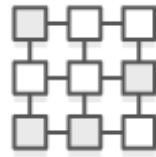
### Build Custom Back-end Services



You can utilize AWS Lambda to make new back-end administrations for your applications that are activated on-request utilizing the Lambda API or custom API endpoints fabricated utilizing Amazon API Gateway. By utilizing Lambda to process custom events as opposed to adjusting these on the customer, you can maintain a strategic distance from customer stage varieties, diminish battery deplete, and empower less demanding updates



## Completely Automated Administration



AWS Lambda deals with all the framework to run your code on highly accessible, fault-tolerant foundation, liberating you to concentrate on building separated back-end administrations. With Lambda, you never need to refresh the basic OS when a fix is discharged, or stress over resizing or including new servers as your utilization develops. AWS Lambda flawlessly sends your code, does all the organization, maintenance, and security fixes, and gives built-in logging and observing through Amazon CloudWatch.

## Built-in Fault Tolerance



Lambda has built-in adaptation to internal failure. AWS Lambda maintains resources over different Availability Zones in every area to help secure your code against singular machine or data center office failure. Both AWS Lambda and the functions running on the administration give unsurprising and dependable operational execution. AWS Lambda is intended to give high accessibility to both the administration itself and for the functions, it works. There are no support windows or planned downtimes.

## Automatic Scaling



AWS Lambda invoked your code just when required and naturally scales to help the rate of approaching requests without expecting you to arrange anything. There is no restriction on the quantity of requests your code can deal with. AWS Lambda commonly begins running your code inside milliseconds of events, and since Lambda scales consequently, the execution remains reliably high as the recurrence of events increments. Since your code is stateless, Lambda can begin the same number of occurrences of it as required without long arrangement and setup delays.

## Run Code in Response to Amazon CloudFront Requests



With Lambda@Edge, you can without much of a stretch run your code across AWS areas internationally, enabling you to react to your end clients at the most minimal idleness. Your code can be activated by Amazon CloudFront occasions, for example, requests for content to or from origin servers and viewers. Transfer your Node.js code to AWS Lambda and Lambda deals with everything required to duplicate, course and scales your code with high accessibility at an AWS area near your end client. You pay for the register time you consume there is no charge when your code isn't running.

## Integrated Security Model



AWS Lambda enables your code to safely get to different AWS benefits through its inherent AWS SDK and coordination with AWS Identity and Access Management (IAM). AWS Lambda runs your code inside a VPC of course. You can alternatively likewise design AWS Lambda to get to resources behind your own VPC, enabling you to use custom security gatherings and system get to control records to give your Lambda functions access to your assets inside a VPC.

## Bring Your Own Code



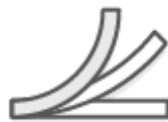
With AWS Lambda, there are no new language, tools, or framework to learn. You can utilize any outsider library, even local ones. AWS Lambda supports Java, Node.js, C#, and Python code, with help for different language coming later on.

## Pay Per Use



With AWS Lambda you pay just for the requests served and the compute time required to run your code. Charging is metered in increments of 100 milliseconds, influencing it to savvy and simple to scale naturally from a couple of requests for every day to thousands every second.

## Flexible Resource Model

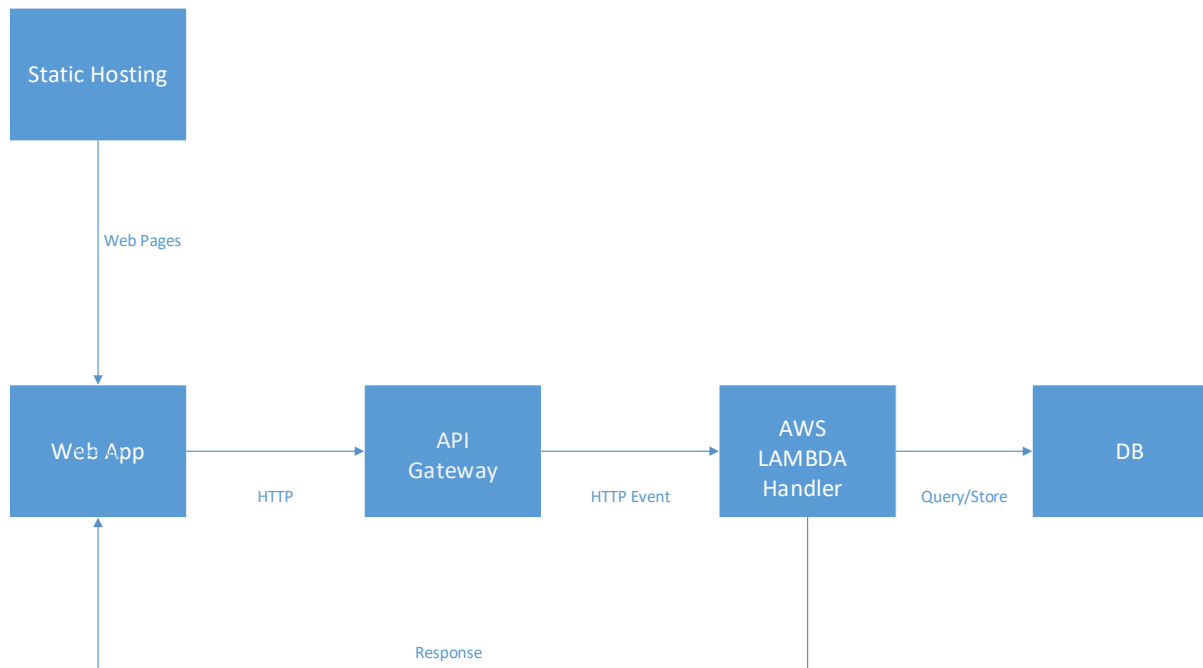


You pick the amount of memory you need to designate to your functions and AWS Lambda allots corresponding CPU power, arrange network bandwidth, and disk I/O.

## Different Use Cases for Serverless Computing or FaaS (Function as a Service):

Let's combine the above building block to something useful:

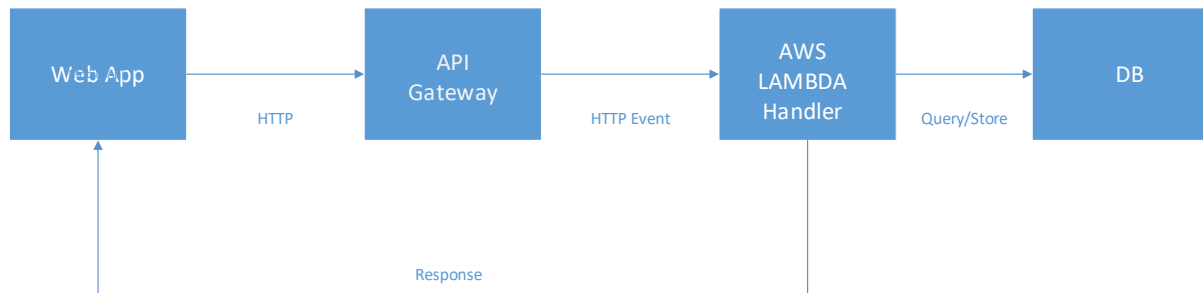
1. Below is the block diagram for the implementation of FaaS using AWS Lambda as a Web Backend.



The use case is a simple web application using the AWS Lambda Handler. The application will present the user with the HTML based user interface. It is interfaced on the backend with a RESTful web service to service the request. The application also provides user to register with the service. The application architecture, as shown above, uses AWS Lambda, Amazon API Gateway, Amazon S3 for static hosting and Amazon Dynamo DB.

- **Static Web Hosting:** Amazon S3 is used to host static web resources such as HTML, CSS, Javascript, and Images that is rendered to the user's browser.
- **Serverless Background:** Amazon Dynamo DB is provided with persistence layer where the data can store by the API's Lambda function
- **RESTful API:** Javascript executed in the browser send and receive data from a browser from public backend API which is built using Amazon Lambda and API Gateway.

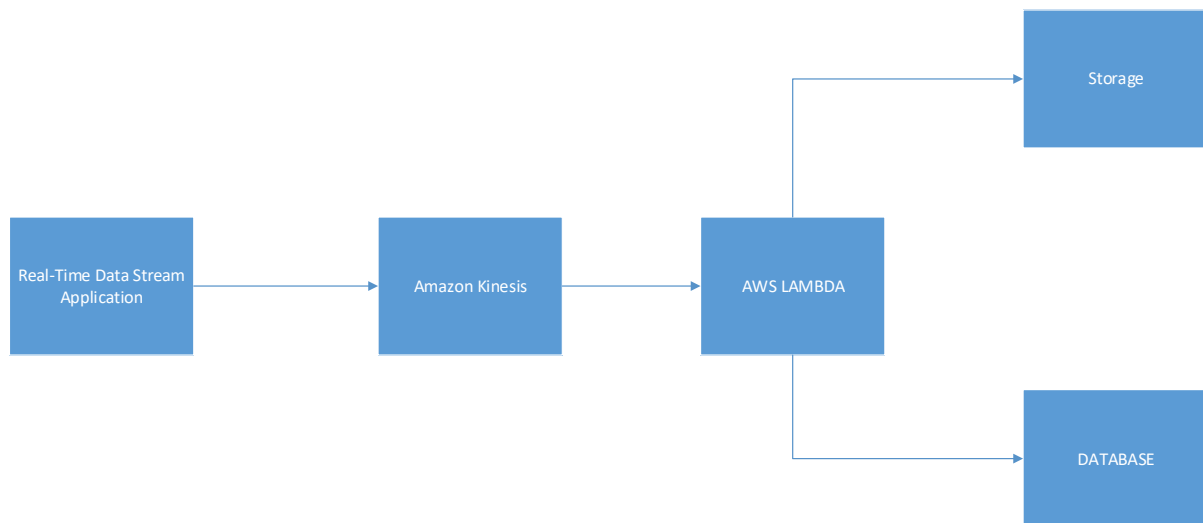
2. Below is the block diagram for the implementation of FaaS using AWS Lambda as a Mobile Backend services.



The above use case is the implementation of mobile backend services using Amazon Lambda, API Gateway, and DynamoDB. Here the REST API calls the Lambda function from API Gateway. Using this API you can perform any operations on DynamoDB from any device enabled with HTTP such as a browser or mobile phones. The client neither requires maintaining any libraries nor maintain any servers. Here API Gateway tells what Lambda function was called. Steps to deploy a mobile backend in Lambda are:

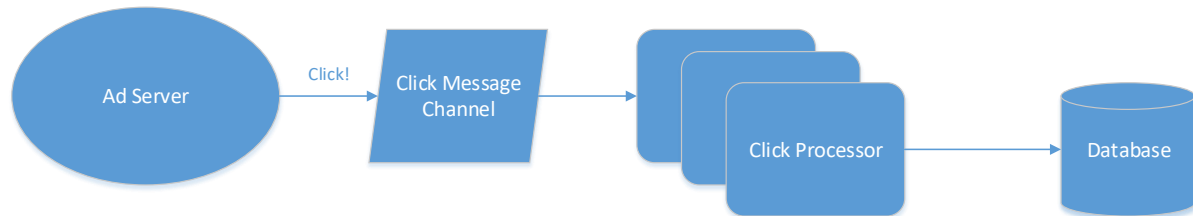
- Set up IAM users and roles to facilitate access to Lambda and Dynamo DB
- Download the sample application and edit the configuration file based on your requirement.
- Create a table in DynamoDB using AWS console.
- Create new Lambda function and upload the file there
- Create endpoints in API Gateway and test the API Gateway and Lambda function.

3. Below is the block diagram for the implementation of FaaS using AWS Lambda for stream processing.



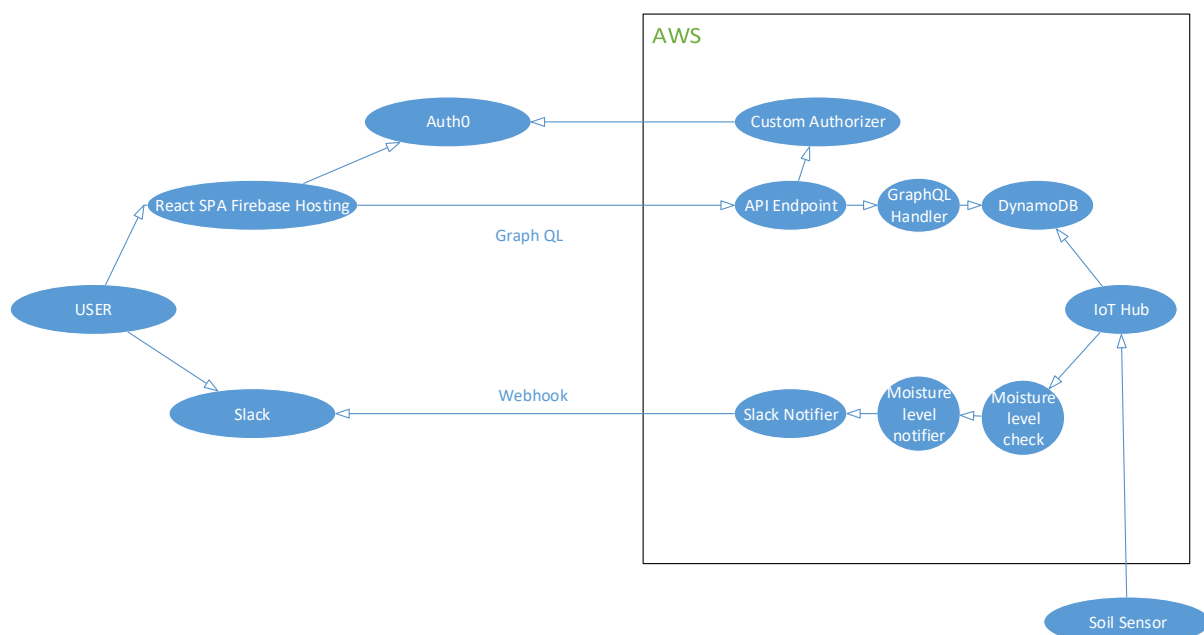
The above use case is used to process stream data such as Log processing which would enable us to run analytics on the stored data. Here the Amazon Kinesis will collect and stream data in ordered, replayable, real-time processing. AWS Lambda is used to create a serverless architecture to execute the uploaded code. Here AWS Lambda runs your code in response to the event such as input Kinesis stream. Here Amazon DynamoDB is used to store the executed stream of data for doing further analytics.

4. The below block diagram for the implementation of FaaS using AWS Lambda as 'Ad Server' click processor application.



Consider the above use case where when the user clicks on any advertisement the system should redirect the user to the target of the advertisement and at the same time the system should maintain a count of how many clicks has happened to the advertisement. So here the click processor is implemented as a function so when the click event occurs the redirect function is called where the user is redirected to the target of the advertisement and click count function is called which decreases the advertisers budget.

5. Below is the block diagram for the implementation of FaaS using AWS Lambda as IoT Backend for garden monitoring system



In the above use case, the IoT Service uses the AWS IoT Device Gateway and Rules Engine. Device Gateway acts as the endpoint for receiving the messages. Rules Engine is used for storing raw messages in Dynamo DB which provides a dashboard and the other purpose is for invoking the check moisture Lambda in the notification service. In the above use case, the notifications service sends a message to slack when the moisture content of the soil is low. Notification service has two lambda functions. The first lambda function checks the moisture level and sends a message when it is too low. This message, in turn, invokes another lambda function which sends a message to slack.

## **Software development challenges using FaaS:**

### **Vendor control:**

With any outsourcing methodology, you are surrendering control of some of your framework to an outsider. Such absence of control may show as framework downtime, an unexpected limit on resources, cost changes, loss of functionality, constrained API updates, etc.

### **Multitenant Problem:**

Multitenant problem refers to various instances of software of different application of different client runs on the same machine and hosting application. This is the arrangement done by the vendor to achieve benefits while scaling multiple applications. But this raises issues such as security concerns, for instance, one client might be able to see other clients code, a decrease in robustness as an error in one client's software might cause other clients software to fail and performance as some client's software using high resources any affect the software performance of another client.

### **Vendor lock-in:**

There might be a possibility that the serverless implementation of different vendors is different. In case if a client needs to switch from one vendor to another the client might also need to change or update the operational tools. In some cases, there might also be a need to change the code or architecture to



comply with the new implementation of the vendor. This means porting clients application is a hassle in case of switching from one vendor to another.

### **Security Concerns:**

The security concerns implemented by the vendor may now be enough and would increase the surface area of the malicious intent and the likelihood of successful attack.

### **Repetition of Client platform logic:**

With serverless implementation, there is no need to write custom logic in the server-side all the custom logic is written on the client side. So as and when the new client gets added all the custom logic must be repeated for the newly implemented client.

### **No Server-side Optimization:**

With the serverless architecture, the ability to optimize the system to improve system performance cannot be performed on server-side. All the implementation must be done on the client side.

## **Implementation Drawbacks of Serverless Computing:**

### **Configuration:**

AWS Lambda functions have no configuration. So, the question is how deployment artifact run with different characteristics. Here you need to redefine deployment artifact with the different embedded config file.

### **Startup Latency**

For JVM implemented AWS Lambda function in AWS the function might take around 10 seconds to startup. Over time AWS will implement certain mitigation technique. So, for now, it is advisable to use JVM Lambdas in certain use cases.

## **Testing**

Integration testing of serverless applications is difficult. Most of the vendors do not provide local implementation but they are forced to use production implementation. This means deploying your code remotely and testing using the remote system for integration and acceptance testing. Part of the problem is that our unit of integration with Serverless FaaS is smaller than with other architecture and therefore rely on integration testing a lot.

## **Deployment**

For FaaS we are missing out of better pattern for bundling up set of functions into the application. You need to deploy dependencies for every function in your application. For instance, if the application is implemented on JVM and you have 10 functions that mean you have deployed the 10 dependencies/JAR. It also means you cannot independently deploy a group of functions. In case of deployment, you need to turn off the source which is triggering the event deploy the group and then turn on the source. This is difficult of application that requires zero downtime. There is no provision for the versioned application so rollback isn't an option. There are open source workarounds but I can be properly resolved only with vendor support.

## **Debugging**

At present, the clients must rely on the debugging facilities that are provided by the vendor. But most of the time we would require open API and third-party services to help.

## Conclusion

Serverless architecture is where we deploy our server-side logic of the application remotely. We implement Function as a Service which moves server-side logic from running long component into short-lived function instances.

Serverless is probably not going to be the right approach for each issue, so be careful about any individual who says it will supplant the greater part of our current models. What's more, be much more cautious if you venture out into Serverless frameworks now, particularly in the FaaS domain. Those advantages shouldn't be immediately rejected however since there are noteworthy positive viewpoints to Serverless Architecture, including diminished operational and advancement costs, simpler operational administration, and decreased the natural effect.

The most vital advantage to me, however, is the diminished input circle of making new application parts since I think there is a considerable measure of significant worth in getting innovation before an end client at the earliest opportunity to get early criticism, and the lessened time-to-showcase that accompanies Serverless fits ideal in with this theory.

Serverless frameworks are still at their outset. There will be many advances in the field over the coming years and it will captivate to perceive how they fit into our building toolbox.

## References

<https://aws.amazon.com/blogs/compute/powering-mobile-backend-services-with-aws-lambda-and-amazon-api-gateway/>

<https://martinfowler.com/articles/serverless.html/>

<http://ieeexplore.ieee.org/document/7980271/>

<https://aws.amazon.com/lambda/>

<https://www.trianz.com/insights/case-future-serverless-computing>

<https://serverless.com/blog/building-a-serverless-garden/>

<https://aws.amazon.com/lambda/details/>