



Expertise  
and insight  
for the future

Bibek Acharya

# Building Serverless Application with AWS Lambda

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

11 May 2020

Author Title	Bibek Acharya Building serverless application with AWS Lambda
Number of Pages Date	45 pages + 0 appendices 11 May 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	IoT and Cloud Computing
Instructor	Erik Pätynen, Principal Lecturer
<p>The purpose of this thesis is to analyze the development of serverless application using AWS Lambda. Another aim was to study serverless architecture with AWS Lambda to build and manage secure serverless applications on AWS. Establishing the evolution of serverless computing along with several cloud vendor's functions can be discovered in this writing. Moreover, the theoretical part presents serverless technology with its history, use-cases, pros, cons, and ultimately the features.</p> <p>The base foundation of the writing is the Lambda function managed by AWS. Employing Lambda as serverless logic, faster, event-driven, cost-effective, secure applications can be built by meeting every compliance concern at every slab. Slightly, touching down the Function as a Service with its vendors, this study clearly shows how AWS has been a success ever since it was launched.</p> <p>As a result of this study, a fully serverless, scalable application was designed and implemented on the serverless database with a new-made API. An effort was made to demonstrate how writing functions, looking at DynamoDB streams, events subsystem, API, and the storage pattern should be carried out. Finally, this study shows the benefits of next generation serverless architecture without bothering to scale application or manage servers.</p>	
Keywords	Serverless Application, AWS, AWS Lambda, API, Python, FaaS

## Contents

### List of Abbreviations

1	Introduction	1
2	Evolution of Computing Architecture	2
2.1	Physical Servers	3
2.2	Virtualization/Virtual Servers	5
2.3	Containers- Late Cloud	7
2.4	Serverless Computing	10
2.4.1	Use-Cases	13
2.4.2	PROS and CONS	16
3	Leading Providers of Function as a Service	18
3.1	AWS Lambda Function	18
3.2	Microsoft Azure Function	19
3.3	Google Cloud Function	20
4	AWS Core Services	21
4.1	AWS Lambda in Depth	22
4.2	Amazon API Gateway	28
4.3	Amazon S3	28
4.4	Amazon DynamoDB	29
4.5	Amazon SNS	29
4.6	Amazon SQS	31
4.7	Amazon CloudFormation	31
5	Project on AWS: Practical Version	34
6	Conclusion	45
	References	46
	Appendices	

## List of Abbreviations

VM	Virtual Machine
OS	Operating System
IoT	Internet of Things
AWS	Amazon Web Services
API	Application Programming Interface
EC2	Elastic Compute Cloud
IAM	Identity and Access Management
VPC	Virtual Private Cloud
S3	Simple Storage Service
NACL	Network Access Control List

## 1 Introduction

The evolution of technology and its infrastructures has always been inevitable. Nowadays, companies are less worried about their IT infrastructures and focused on their key target by virtue of the cloud. In the software architecture world, serverless is a hot topic. The top three vendors- Amazon, Google, and Microsoft had been investing loads of resources for the advancement of the serverless environment. Serverless computing has been gaining momentum due to the gradual development of the cloud and insufficient manpower to manage and compute infrastructures or containers. Cloud vendors manage IT infrastructures and users pay as they use, which leads serverless as an emerging cloud computing model. It can be also stated as the next layer of abstraction in cloud computing. [1,2]

This study will depict serverless computing and its evolution. Starting with the monolith era, virtualization, containers, and serverless era, the thesis continued by presenting top vendors in the market. Further, preferred vendors for the project are AWS and its Function's AWS Lambda was explored deeply along with the services that can be integrated to create a serverless environment. The project will carry out on the AWS free tier platform, implementing its several services with AWS Lambda, creating and approach the best practice to write the Lambda Function and deploying and testing serverless Applications. Overall, building and designing a cost-effective and highly scalable serverless application using AWS Lambda Function is challenging. One should have a clear vision of the architect when building a serverless application on AWS, as this final year project clearly shows.

Structured with the five-section, the first section introduces to the subject matter of the thesis. The second section mentions various eras of computing with the pros and cons of each. Similarly, the third section mentions detailed information of the top vendors of serverless computing after describing the Function as a Service. The fourth part focuses on implementing AWS and Lambda Function along with the different services integrating with it. The final section presents how a practical version of serverless application was carried out and presents the outcomes of the project.

## 2 Evolution of Computing Architecture

Evidently serverless is the next logical path on the advancement of cloud computing. Starting with bare metal following with the virtual machines and containers, the computing had accomplished to the era of serverless where one does not crave about the server and its maintenance. Adopting serverless architecture, highly flexible, and stateless applications can be created for a variety of industries. Figure (1) portrays the gradual progression of the distinct era with history of IT computing. On top of that, below clause will incircle the graph in a systematic order.

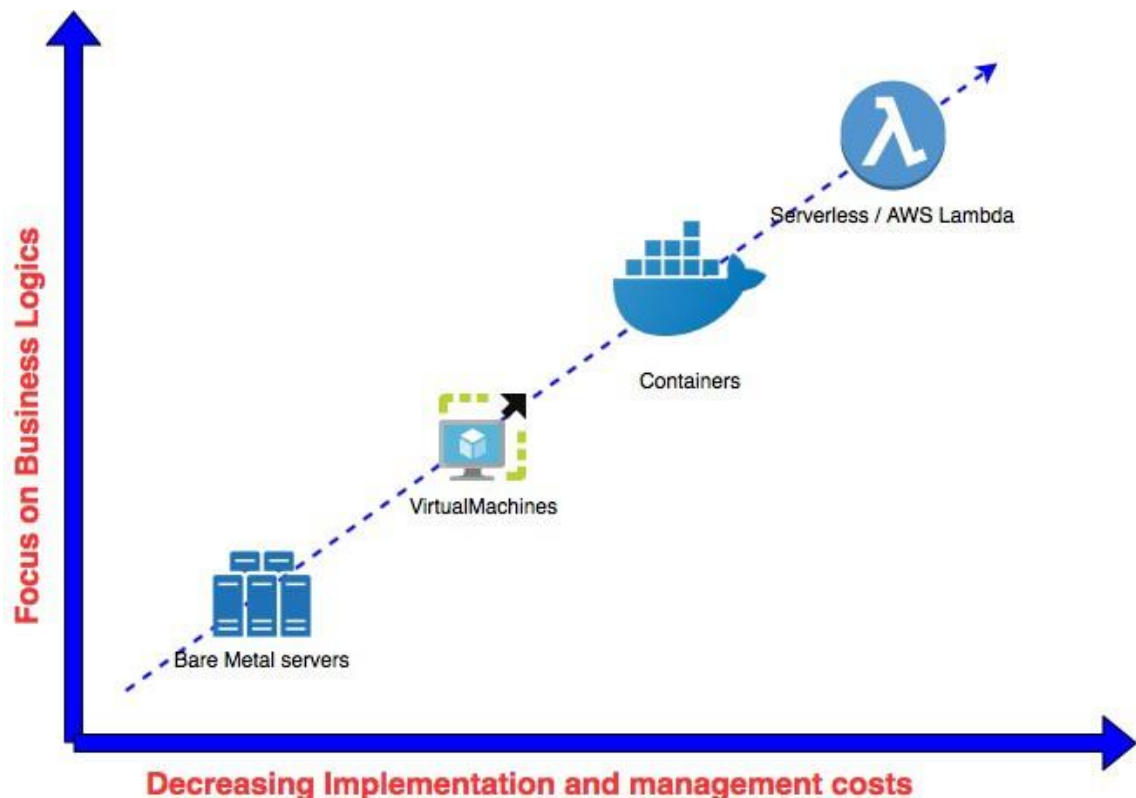


Figure 1. Evolution of Computing Architecture [3,1]

## 2.1 Physical Servers

Originally, we had begun our journey with physical servers, which required stacking and racking of the big physical boxes and installing an operating system on it. In other words, Rack'em and stack'em. The design of physical servers is quite simple and consists of network, processing power, memory with storage capability. The servers with installed operating system assist in running applications. [2,1]

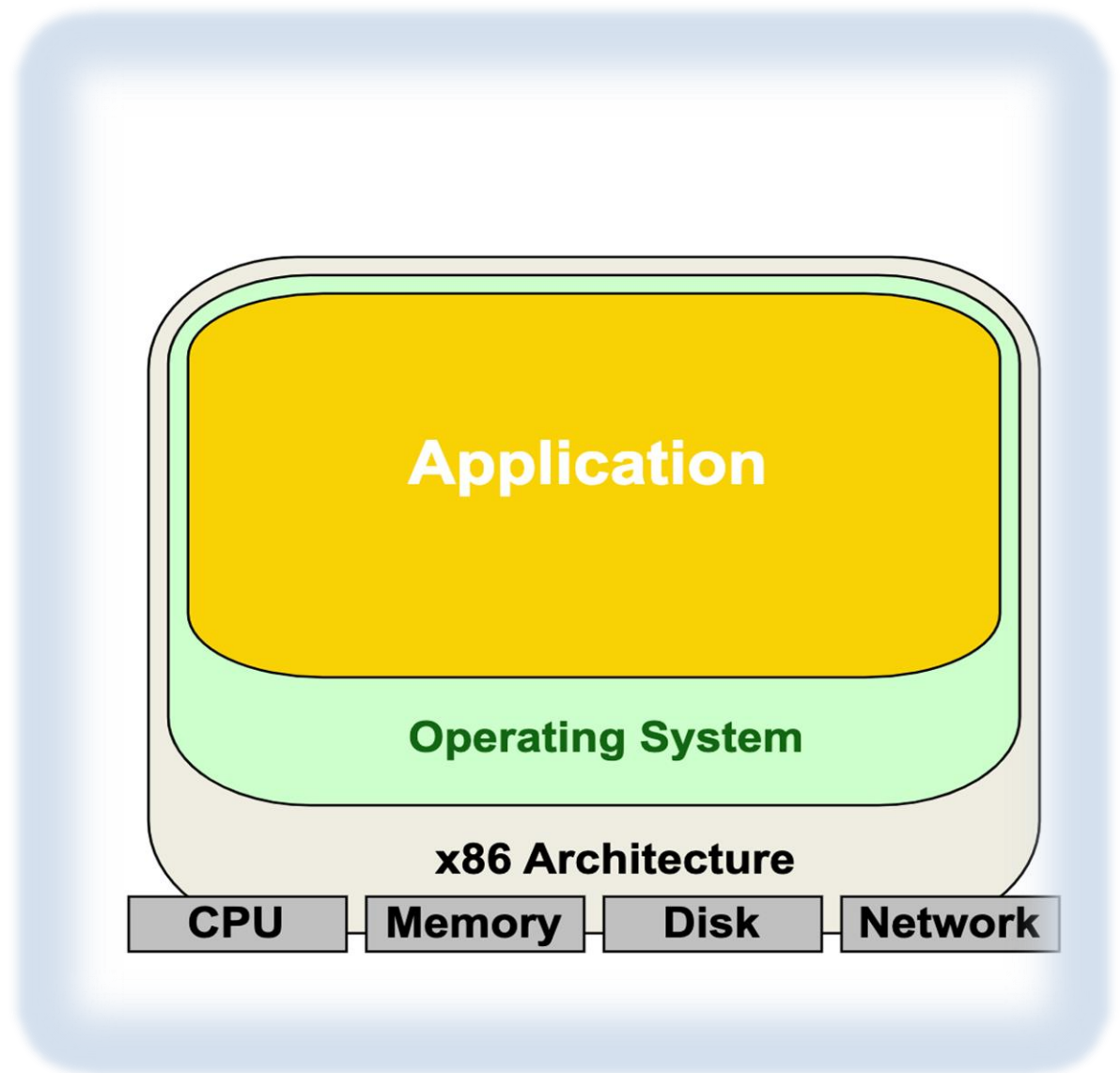


Figure 2. Traditional Physical Server- 1Physical layer, 1 Application

As shown in figure 2, traditional physical servers consist of one physical layer and application. Each includes memory, processor, disk, network, and operating system. Supplying adequate power supply, cooling system, and high-speed networks inside the data centers would require for those giant boxes. It can be also referred to as bare-metal servers or managed dedicated servers which are allocated for the single user where the owner is only with the server access. These arrangements will not stop after installing an application, dedicated staff would require for administration and maintenance.

### **Pros and Cons of Physical Servers**

Generally, physical servers are effective for high-level traffic which comes up with accuracy and a dedicative hosting environment. As IT staff has direct access to the system, quick responding and troubleshooting could be performed lessening the downtime of the servers. Further, servers perhaps design and customize as per business needs accordingly. Even today, it is in practical use for certain business-like cryptocurrency mining under latency and easier customization of physical hardware as per their needs.

Apparently, high cost is the root cause of replacing traditional physical servers where one not only has to buy all the hardware part, again maintain and upgrade the system time to time, the result in costing the resources indefinitely. A separate space and energy resources must be allocated depending upon the size of the company. Sometimes, Failure to hardware results downtime of the servers since replacement parts needs to be ordered. [4,1]



## 2.2 Virtualization/Virtual Servers

Looking back to the 1960s when most enterprises had physical servers with single-tenant access that only allows running applications on particular hardware. Enterprises had to make a change to virtualization to embrace the partition of the servers and multi-operating environment. Eventually, the term virtualization came to popularity and adopted owing to its features and uniqueness. Moving on with the minimum cost of hardware, cooling system, and maintenance, utmost of the companies updated their IT structures to virtual that help vendor to utilize their resources inexpensively wise.

The virtualization refers to the practice that utilizes the software for creating the virtual version of the application, servers, storage, and networks where the different operating system is being run in a single computer and acted as multiple virtual computers. Here virtual computers are virtual machines that simulate a physical compute in software form created by a program called a hypervisor. Hypervisor acts as an interface between physical components and VMs. Further, it is classified into two categories called as 'bare-metal hypervisor' and 'hosted hypervisors'. [5,1]

### **How Does Virtualization Works?**

Figure 3 clarifies the hypervisor differentiates the physical hardware apart from virtual environments that is a software layer of virtualization architecture. It's on top of the operating system or straightly installed onto the hardware alike server. Following detachment, resources are divided as per the requirement to many virtual environments later, system users work on compute with VMs. Conclusively, users can ping the set of instructions that requires extra resources from physical environments once virtualization is up and running. The virtual machine behaves as a single data file that can be transferred and run into several computers. Thus, possessing the characteristics of partitioning, isolation, encapsulation, and hardware independence a large number of enterprises had been adopting it to score generous outcomes with minimum resources. [5,1]

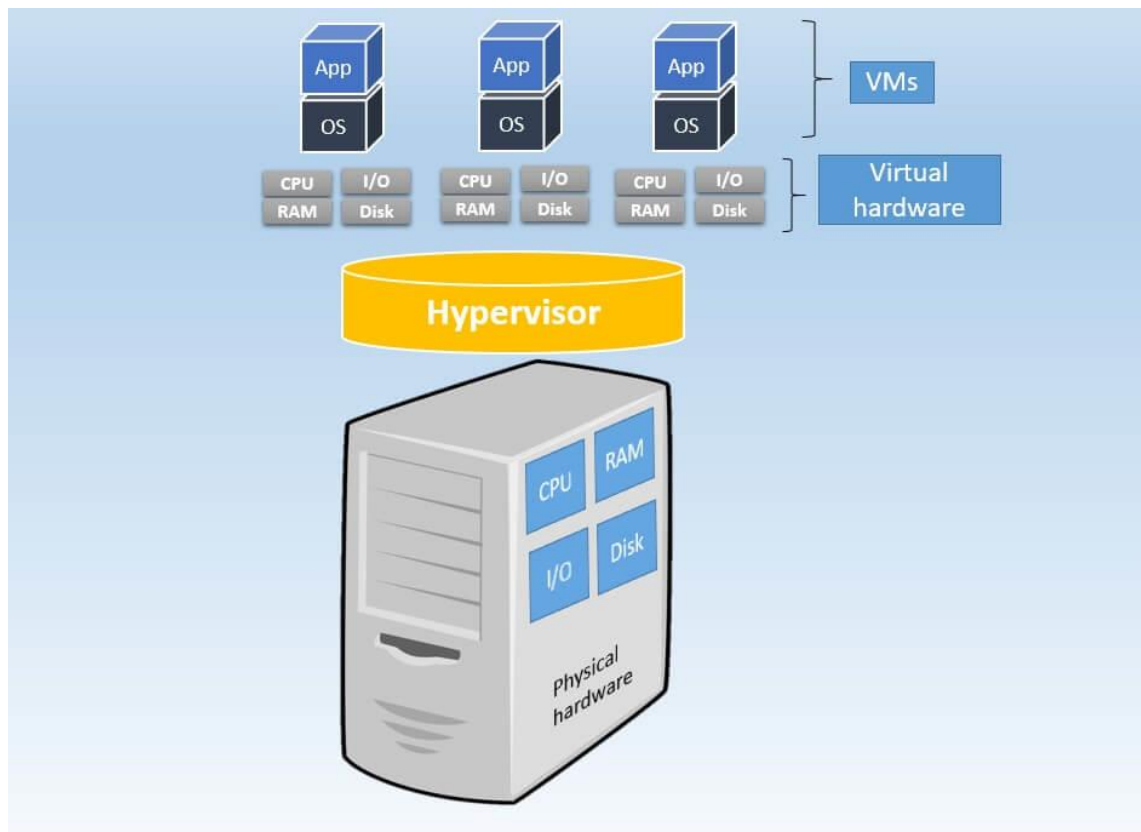


Figure 3. Components of Virtual Environment [1,1]

### Benefits and Challenges of Virtualization

Alike other technologies, virtualization also comes up with pros and cons that ought to be considered before implementation. The necessity to have physical machines, dedicated space is swiped out by virtualization. Owning the license and access from the vendor is enough to get started with, ultimately cheaper and timesaving for the company. The requirement of infrastructures and cost can be readily forecasted. Further, it diminishes the workload by improving run time. Thus, increasing IT agility, security, scalability assist the enterprises to achieve their requirement. [6,1]

Possessing many advantages, it does have part of slight challenges, needs to consider before adopting it. It seems to have a lower cost on the perspective of the user despite would require hardware and software for providers as implementation price could be steep. Lagging and availability are the next challenges might occur as many VMs are running under a single resource. Periodically, the installation of servers consumes time

compare to physical servers. [6,1] Nonetheless, cons can be the outcome if it's implemented an inadequate approach.

### 2.3 Containers- Late Cloud

One of the trendy topics of the cloud computing field, containers. It is one step ahead of virtualization, a technique of operating system that permit to run app and dependencies related to it in resources- isolated development. Unlike virtualization, it favors installing multiple apps in a single Operating system that means supports OS-level virtualization, abstracting 'user-space'. Enhancing the framework performance, it allows the control over resources.

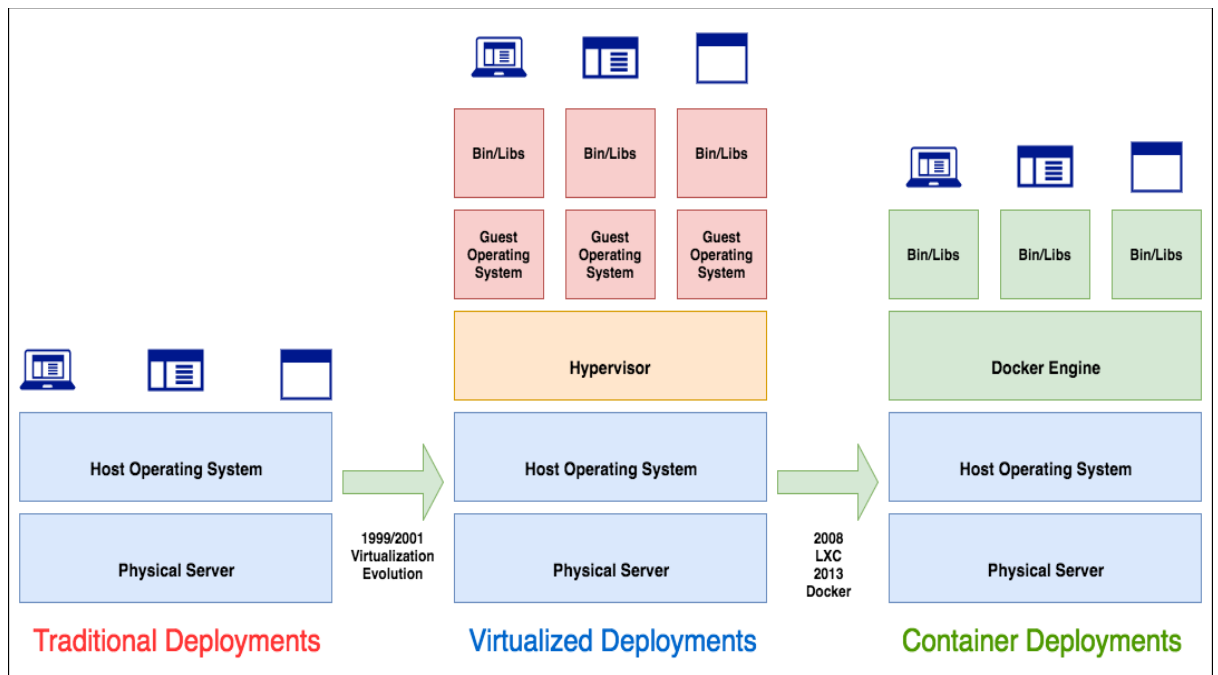


Figure 4. History of Application Deployments [7,1]

Figure 4 interprets the improvement of the deployments over a single server, app within the distinct OS, and many applications in a single operating system. Containers occupied less space with low computing power compare to virtual machines. Despite VMs intent are alike to containers, sharing the host systems kernel to other containers make them diverse. [8,1]

## How does Container Work?

Before talking about the working process of containers, let's walk through the docker. Docker is the tool that is designed to create, deploy, and run app adopting containers. It is based on Linux containers that is open-source project, implement several features of Kernel to create OS. [8] Namespace, control groups, and union filesystem are the features that assist to picture the working process of containers. Where namespace implements along with MNT and USER namespace that define filesystems and user, groups IDs respectively. Similarly, control groups ensure the right amount of CPU, memory, and network containers need. The union file system is used to help Lessing the duplication of data, created each time. Repositories, container API and container creation assist to run and transfer the images. [9,1]

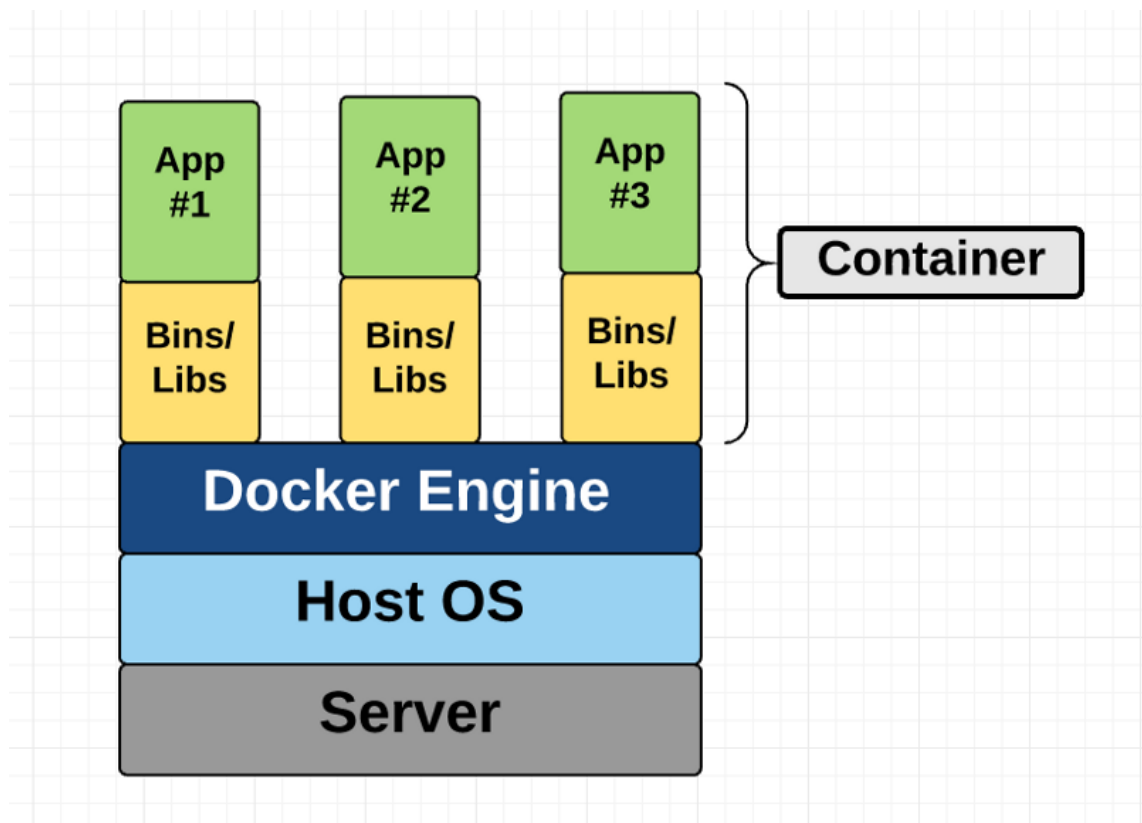


Figure 5. Diagram of Container [8,1]

As shown in figure 5, the OS-level architecture is only shared, and bins and libs are created from scratch.

To continue with the working procedure of container, Docker and Kubernetes are the key platforms to manage the containers despite having different working nature. The following commands display the distinct method of running the same file.

```
- def Nginx container running with Docker command
- $ docker run -d --restart=always --name nginx-app -p 80:80 nginx
  And the Nginx container running with Kubernetes (Kubect1) command
- $ kubect1 run --image=nginx nginx-app --port=80 -- env="DO-
MAIN=cluster"
```

Listing 1- Command for Running File on Container [9,1]

## 2.4 Serverless Computing

Serverless computing commonly quoted as the next generation of cloud computing. The term serverless itself is baffling as most of a new people to it might judge as being practice without any servers to host and run the code. Alternatively, it refers to the approach in which end-user exclusively focus on their business tasks and model instead of spending time and money on server status, maintenance, scaling, capacity, and OS patching. All in all, it is defined as the practice of building a running application without managing servers. Even though, the cloud vendors still must manage servers, VMs, containers, and Operating system that is abstracted from the developer. Writing code and creating a function that is even triggered is done by the developer and executing it with the right infrastructure is handled by providers. Moreover, the developer can select the right tools and languages as it comes with different options. [10,1]

The simple architecture on serverless is shown in figure 6. The function can be deployed and tested separately. Users would only pay for the active functions while stand-by functions only use sources if needed. For instance, 'create user' is requested by the user, only 'create\_user.js' will be executed while all other functions will be on standalone and run if needed. [10,2]

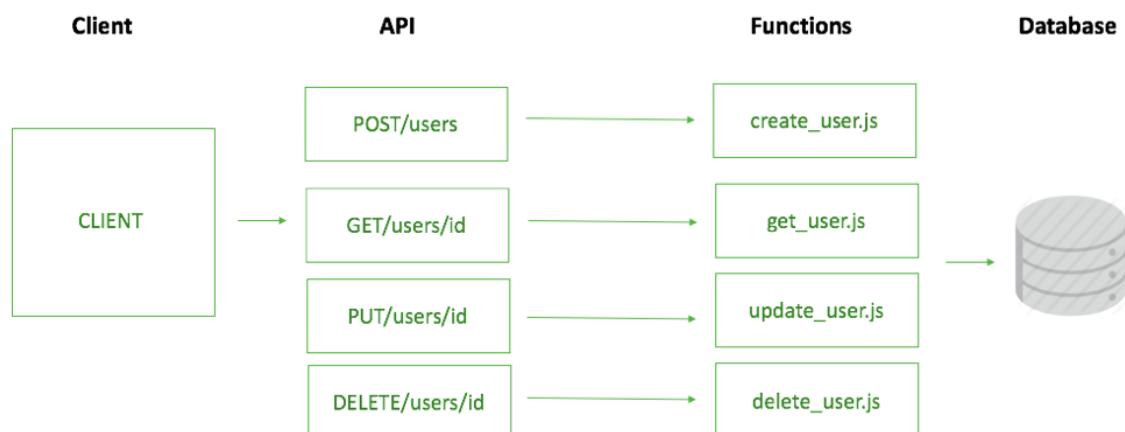


Figure 6. Serverless Architecture [10,1]

## Origin and Growth of Serverless Computing

Serverless computing encircles with distinct however overlapping areas and that are included by cloud providers into their service model. Function as a Service (FaaS) and Backend as a Service (BaaS) where the function is triggered in event-driven format and API from third-party that displace sets of functionalities respectively. [11,1] Thus, the development of application on serverless is solely dependent on third-party API, client-side logic and cloud-hosted service known as FaaS are transcendental from the developer, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Container as a Service (CaaS) and Function as a Service (FaaS). Outsourcing the limitation of these services, serverless computing embraces FaaS and becomes more popular as AWS introduced AWS Lambda in 2014, following with the release of API gateways in 2015 and by mid-2016, the term serverless ended of being a most dominant topic in a particular area. Following the release of the Lambda function, Microsoft and Google published their closer function naming 'Azure Function' and 'Cloud Function' severally in 2016. However, the first company to provide service was Hook.io in October 2014. [12,2]

The figure 7 below, depicts the growth of serverless computing from the beginning phase and flourishing interest of media and adoption of the serverless that is likely to be numbering even in the future. The Attitude of competition among cloud providers is pouring the flow of services and its popularity in serverless computing. CB Insights Market Sizing tool predicts the value of 7.72B dollar by 2021 with a 33% of annual growth. Hence, due to the logic behind the serverless technology not only has it been progressing economically, as well have earned popularity in adoption by the number of enterprises along with the character of distributed computing. [13,1]



Figure 7. Growth of Serverless Computing [13,1]

### Function as a Service (FaaS)- The Backbone of Serverless Computing

Function as a Service abbreviated as FaaS another name of serverless computing is among the services of cloud that tends to allocate the service to develop, run, and manage the application functionalities in the absence of infrastructures. Developers all need to do is write a logical code that responds to various events, concentrate on the development of the application, and let the platform take all other key responsibilities- like passing a bunch of information from one function to another, triggering logic of function, scaling, and management. Functions are much alike as microservices where microservices is an approach for developing software system with set of small services, running in its own methods. In microservices, isolating huge monolith code into manageable elements, it updates and scales those elements accordingly. On the other hand, FaaS endure it to another level by breaking monolith even farther. Ensuing the better knowledge of Function as a Service, one should have a better understanding of the major components of it,

- Functions- It is an independent unit of deployment that is primarily responsible for a certain task as of processing files, saving users to the database, operating a scheduled task.



- Events- whatsoever triggered from the functions is considered as an event. It could be the publishing of the message, uploading of files.
- Resource- The components or services utilized by functions to access convincing results. Database services, File system services as in storing data, Message Queue Services for broadcasting messages. [12,2]

Certainly, its widely different from PaaS, where deployed application runs on one server all the time.

#### 2.4.1 Use-Cases

Real-world has got a diverse source of application from the aid of serverless computing. The interest to implement Serverless is determined however with another non-functional area such as cost, amount of control over needed operations, and workload. Huge companies like Netflix, Coca-Cola, Expedia had been implementing serverless computing to enhance their services. Event-driven and flow alike processing sequence carrying out in image and video processing, consumption of API, serving static content are some instances of real-world application and are discussed further below,

Event- Processing: In other words, event-based programming where files are processed for analyzing data. For large applications, there will be always imaging, videos, log-files that has to be processed. The event-based function that is integrated into serverless computing is the perfect fit. Figure 8 (below) implicates how the video files are being processed implementing the serverless function. Videos are first uploaded to the storage system, which ejects events that trigger the function leading to isolation of the video and transcode again to another format. The function could be executed again without any harm in terms of failure as it is stateless. Other sophisticated applications such as chat-bots, stream processing, and web application can be developed implementing serverless functions. [14,16]

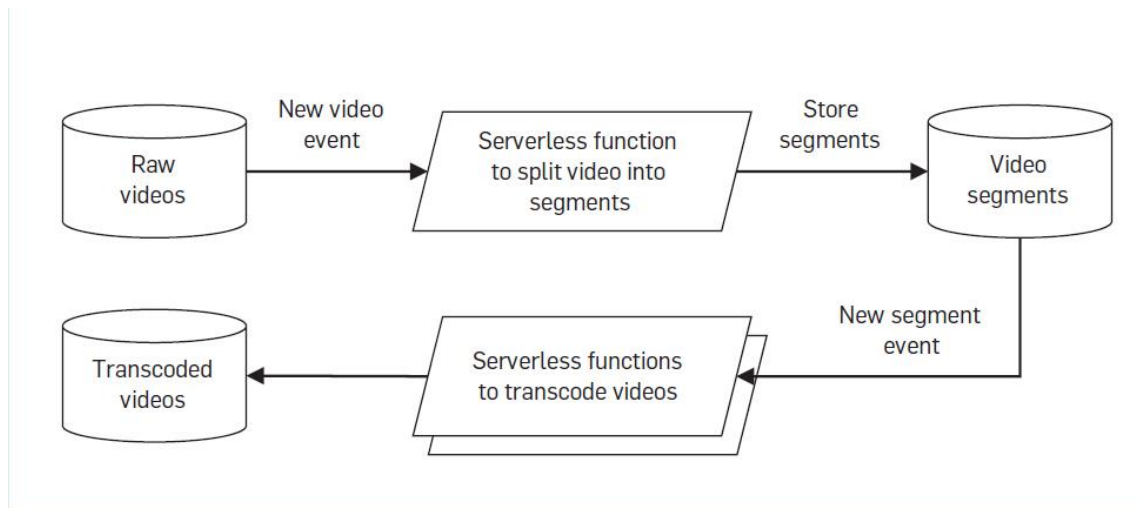


Figure 8. Video Processing [14,16]

API Composition- Application Programming Interface (API) is a medium of communication between two different software applications and implementing a query mentioning API composer, data can be invoked within the services. Normally, API gateway performs API composition. Here, data filtering and transformation are included in the application. Figure 9 is a prototype of mobile app that invokes geo-location, weather, and language translation APIs to publish the required information about the weather forecast of customers' current location. In serverless function, the source code can be defined as in picture it's defined in python language. Thus, the cost of invoking several APIs in a restricted network could be skipped or performed by the defined source code that is the function of serverless computing. [14,17]

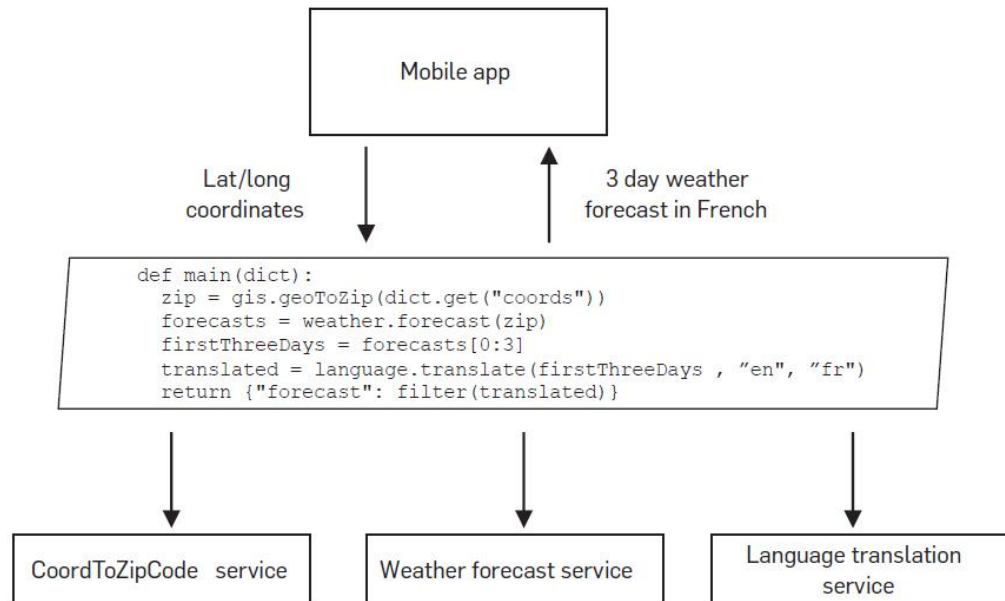


Figure 9. Migrating API calls and logic to backend from mobile app [14,17]

Multi-tenant Cloud Service is the next application where the company grants the services on-demand by leveraging cloud services and serverless computing. Multi-tenant, high available, secure, and scalable with a serverless function can be built. Similarly, business logic, API aggregation to reduce API calls, Agile and continuous integration pipeline, virtual assistants are a few other applicable scenarios for the serverless computing.

## 2.4.2 PROS and CONS

Undeniably, one should analyze the advantages and downsides of serverless before approaching to implement serverless computing. As it allows the developer to get their hands on the design and code of the software with functionality of the scalable and secure environment. Listed below are the advantages to be discussed further, why serverless is often a good choice.

- No infrastructures to manage- Along with the aid of a serverless platform, the developer can instantly write and deploy the code beyond concerning the hardware, operating system, and servers. As managing and patching of servers are taken care by vendors whereas user can save the money and time.
- No costs when functions are idle- It is incredible, how the technology had been transformed over the year. Once, we used to have servers running every hour and spend our money even though in an idle period. However, with serverless function, you do not have to pay anything if it's not running at all.
- Multitude Uses-As Parallel processing could be achieved through two or more computers between the network that aid the compute to be stateless and scalable. Adopting serverless architecture, all sorts of mobile and desktop software, as well as backends for e-commerce, CRUD applications, web apps, can be readily assembled. Startups wanted to grow and innovate, would also approach serverless within viable cost. [15]
- Exceptional cost- The substantial advantages are low cost as one only pays for computing power used. With the several functions use by various cloud providers, you only pay for the services that are being used. Even during the peak, heavy and unexcepted traffic, the service run in a scalable way with a significant price that leads to the huge economic win for the company.
- High availability and scalability- Lambda function from AWS along with functions of azure and google cloud, capabilities for automatic scaling is worth spending as per our needs accordingly. Similarly, the partition of a new cloud server, the

purchase of more computational power, composing highly available applications are taken care of by serverless platforms.

Reduced latency, software complexity with the improved user experience are the other benefits that can be achieved from it. Still, serverless computing is not a magic bullet in every situation, vendor lock-in, decentralization service, security issues are in the path hindering to use it. The following points are included in the drawbacks.

- Vendor control- All the system is upon the control of third-party vendor as a result, system downtime, loss of functionality, forced API upgrades could be issued. Similarly, data sovereignty, privacy, cost, the viability of a company may be at the point of risk due to dependency all upon vendors and have to play with the rules from the cloud provider.
- Customization- Assume, you have been practicing the services from AWS with a wide variety of customization but suddenly happen to change to other providers, then the level of customization would be different levels, and porting your application results in a complicated situation.
- Decentralization- Despite the availability of extensive resources and guides, still shifting from monolithic practice to decentralized serverless is challenging and steep whereas splitting from a monolith into microservices is even complex. Due to the distributed nature of the solution, one might have to take help from professionals. [15]

Moreover, complexity to debug and handle stateless function, needs of separate tools and IDE might limit the adoption of serverless in certain cases. In a nutshell, Software developers have to think twice before diving into serverless computing, minimizing the complexity and limitation for the enrichment of their organizations.

### 3 Leading Providers of Function as a Service

Countless providers of FaaS aka serverless computing have arisen in the market while keeping all besides AWS Lambda, Microsoft Azure and Alphabet's Google Cloud platform are the top-notch providers with dominance. Of course, assorted elements make these platforms distinct to each other. To have a better understanding of these functions, one should study them and analyze the differences. The next chapter will illuminate how different functions can be created in these platforms with their characteristics.

#### 3.1 AWS Lambda Function

Amazon Web service was pioneer for pitching the serverless computing via Lambda product. AWS Lambda is a serverless compute service that is responsible to run code in response to events and handle the computing resources. 'Lambda Function' is the name where the code runs accepting single variable input and implement to describe logic in an easier way. To cope up with the scale of incoming events, several copies of functions are required and to perform this AWS Lambda must be written in a 'stateless' pattern. The highly scalable and secure serverless application can be built with Lambda integrating with Amazon VPC, Amazon API Gateway, IAM, S3, Amazon Kinesis stream, Amazon SNS notification, etc.(12) Whenever a file or video is uploaded in S3, there would be Lambda function waiting to be triggered for the process to be complete and in return can trigger other Lambda operations. Amazon provides the number of choices to configure and optimize which results in complexity in writing the Lambda Function. Developers got choices of programming languages to write it in JavaScript (Node.js), Java, C#, Python, and still with many other languages like Lisp, C++. [16,1]

#### Creating Simple Lambda Function on AWS Free Tier

Before writing the Lambda function, it does have a certain condition that ought to be full filled. The first point is to provide a handler that is considered to be an entry point of Lambda. Secondly, runtime environment must be specified, basically run time is a language to write the function and finally requirement is a trigger which is a code that will respond to every event in DynamoDB streams or in S3 bucket. Knowing components Lambda function below as in picture 10, a simple lambda function was created to test

the sum, differences, product, and quotient of two numbers. The test event was configured with two numbers as, {"Number1": 10,"Number2": 20}. Here a sample Lambda function was created from scratch that was pretty simple and straight forward to trigger an event and test it . The Python code written to create the function is clearly seen in the screenshot below (Figure 10).

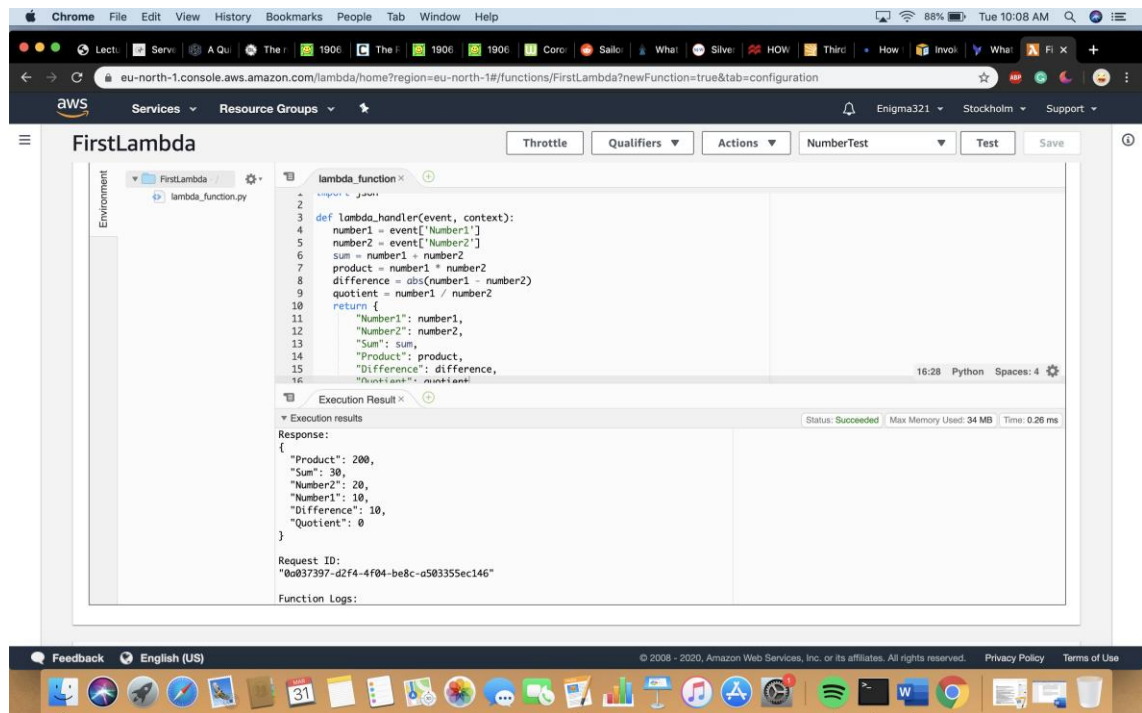


Figure 10. Lambda Function on Python (own AWS account)

### 3.2 Microsoft Azure Function

Similar, to AWS, Microsoft Azure is another competitor in the field of cloud computing. The function written for serverless computing here is called an Azure Function that provides a measure to write code and run it on demand. Few services and even third-party services can be integrated with Azure Functions. Azure notifications, Service bus, Document services are few of the services implemented either triggering the function or handling as input and output for Azure Functions. Allowing a user to develop serverless applications, an option of plenty of choices for language like C#, Java, JavaScript, Python and PowerShell, Pay-per-use pricing design, integrated security is some of the key

features of Azure Functions. Additionally, it is the finest solution for data processing, Internet of things(IoT), microservices, building APIs, image processing, file maintenance, etc. (12.Logic App here encodes the workflows, comparatively complex to Functions. It is user-friendly, the user with semi-programming language knowledge could understand and use it. Logic Apps do have pre-built 'connectors' which is feasible to connect bigger Microsoft and third-party apps. [16,1-2]

### 3.3 Google Cloud Function

Google Cloud platform proposed the Google cloud function, where developers can create functions that acknowledge to cloud events regardless of managing server and runtime environment. It is carried out within the Node.js runtime environment. Converting uploaded docs into pdf or image to thumbnails in google cloud storage are few use case scenarios of google cloud functions. [12,3] Likewise, if an organization aims to get away with the configuration of the server and develop a full stack serverless application, google cloud function could be one viable option. Google had been calmly adding up serverless ideas within their services. Google Cloud Pub/Sub, Google Cloud Functions, and Google Firebase are some of them. You only pay for the compute power usage as Cloud Function scales up and down. Through google cloud, end to end complex development can be built with attached security at role and function level. It has the features of monitoring integrated with it, flexible tracing and network capabilities as well.



## 4 AWS Core Services

Earlier, analyzing the top vendors in the area of FaaS, AWS is considered to be one of the finest providers in the market with diversifying the services and foreseeing the brilliant future. Over and above, the paper will deep dive into AWS and services equipped for a viable development of the serverless application. Amazon Web Service (AWS) is a subsidiary of Amazon Inc, a leading cloud platform in today's generation to consider as a tremendous invention to deploy various kinds of applications to the cloud. The extensive set of global-based products with computing, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security, and so on are the services afforded by AWS. In 2006, AWS commenced providing IT infrastructure services to business as a web service that is now commonly known as cloud computing. It does not stop right there, navigating with many obstacles and challenges, AWS offers a highly reliable, scalable, low-costing platforms in around 190 countries supporting their business in an advance way.

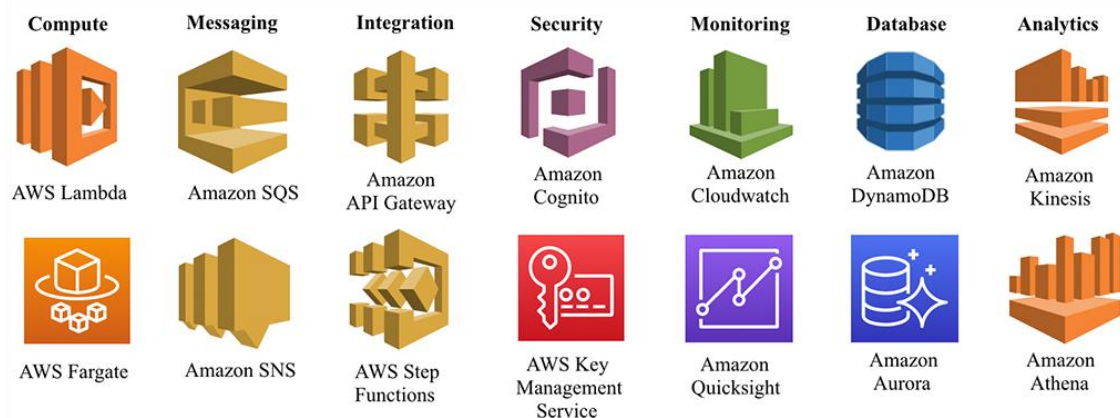


Figure 11. Services in the AWS Serverless Platform [17,1]

Acknowledging the serverless world, services shown in the above figure 11, are integrated with the pursuance of building the application in a dynamic and flexible approach. The AWS platform also encompasses a set of developer tools like Serverless Application Model (SAM) which streamlines the serverless application deployment. Thus, eliminating infrastructure management, AWS grants a diverse range of cloud services including solutions such as compute resources, storage types, big data stream processing, messaging and monitoring services, machine learning, and so on. Further writing will interpret

an overview of core services that can be integrated with each other to provide dynamic and powerful serverless applications. [17,1]

#### 4.1 AWS Lambda in Depth

AWS Lambda is the heart of serverless, an event-driven serverless computing platform on AWS. It was introduced in 2014, at the yearly AWS re: Invent conference in Las Vegas. The core concept is pretty much clean where Lambda runs developers' codes in response to specified events, scales automatically, and provides built-in Amazon watch code monitoring and logging. In the beginning, AWS invented Lambda to clarify the certain issue of EC2 that is to respond and handle events. Even though EC2 is a widely accepted service of AWS, having the distinct feature Lambda is being largely famed. Every Lambda function created contains the code to execute, configuration to mention how the code will be deployed and event sources to observe events and invoke the functions. The simple mechanism of running lambda function can be seen in figure 12 below. While working with Lambda, usually have to deal with Lambda function as a developer. In the process of creating function- mentioning the permissions for the function, specifying events to be triggered, giving code, libraries, and configuring the executing parameters are handles by the developer. As the function is invoked, Lambda does afford the execution environment placed on runtime and configuration selected before that is AWS Management console. [18,2-3]



Figure 12. Running Lambda Function [18,4]

## Key Features

Lambda allows the implementation of serverless and microservices programming architectures to reinforce Function as a Service (FaaS), assisting to run and execute backend code. It does acquire discrete characters, outlined below:

- **Bring your own code-** Users do not have to get into the new languages, tools, or framework. The Lambda Function can be bundled with any other libraries, Linux executable files, or even native ones. For future scenarios, it can be called by such executables and the user just need to upload the code in the format of zip or jar archive and Lambda handles the rest. Java, Go, PowerShell, Node.js, C#, Python are braced by it along with the Runtime API.
- **Integrates with and extends other AWS Services-** Everything that is done in traditional application along with calling AWS SDK or invoking a third-party API can be done by Lambda Function. The custom logic to AWS resources like AWS S3 buckets and Amazon DynamoDB tables is allowed by it.
- **Availability and built-in fault tolerance-** Without and additional configuration, high availability and fault tolerance are baked into the service. To avoid the machine and Data Centre failover, Lambda manages the computing capacity across the multiple Availability Zones.
- **Don't pay for idle-** With the lambda, one is charged for the duration of Invocations and invocation requests. Instead of paying per server, only for execution duration is billed and never have to pay for idle capacity. Whenever functions run, an invocation running for 2 seconds would cost twice as much as an invocation that runs for 1 second but durations are charged in 100ms blocks.
- **Security-** The code written is securely accessed to other AWS services over its built-in SDKs and integrated with IAM and ran within VPC by default. More than that AWS Lambda let you leverage custom security groups and network access control list (NACL) in regard to connect other resources.

Likewise, orchestrate multiple functions, flexible resources, permission, and concurrency models further are some of the other remarkable features. [21,1]

### How Lambda Works?

Individual Lambda function runs in its own container. Whenever a new function is created, it is contained into a new container and it implements the same container on a multi-tenant cluster machine operated by AWS. An appropriate amount of RAM and CPU along with the time frame it requires to execute is specified while the function starts running. The RAM can be configured from 128MB to 3008MB, in 64Mb increments and timeouts of 900 secs. Assuming that, the function fails to outright processing by given timeouts, Lambda will time it out and display an error message. To be precise, users are charged as per the memory used and the run time taken by function to finish. As the updating machines, stabilizing the network and the absolute infrastructures are handled by AWS, customers barely know how the system works. That off course assists users to provide more time and hands to develop and focus on their application code. One of the features of AWS Lambda is concurrence where several instances of the same or different functions can be run simultaneously. [22,1]

The handler or the code execution starts as the Lambda function is invoked and is the code method that's been created and comprised of the package. It can be either on Java, C#, Node.js, or in python with its mandatory definition. The handler can be written as `def (handler_name (event, context):` and return to a certain value on python:

Once the handler is invoked, the Lambda function is ready to run any logic, follow up by the code written in the handler. Further, the handler can call different methods, functions, third-party libraries, and even connect with other AWS services. The event object is one of the parameters for the handler function where all the data and metadata to push the logic is stored. Likewise, the Lambda function is provided with a context object that establishes possible interaction between function code and Lambda functions. More or less the context object consists of AWS RequestId, Remaining time, and logging-ability to stream log statements to Amazon CloudWatch. Besides it is meaningful to know that AWS Lambda is a stateless service, meaning the written code could not make any assumptions about the state as function created and invoked for the first time are

thoroughly managed by Lambda. However, after completing execution, before being terminated the container remains available for a few minutes termed as cold start and in case the same possible function and container is invoked by AWS to initiate a new call, such action of deploying active function containers is referred as warm container and gain the response time of Lambda accordingly as revealed in figure 13. [18,]

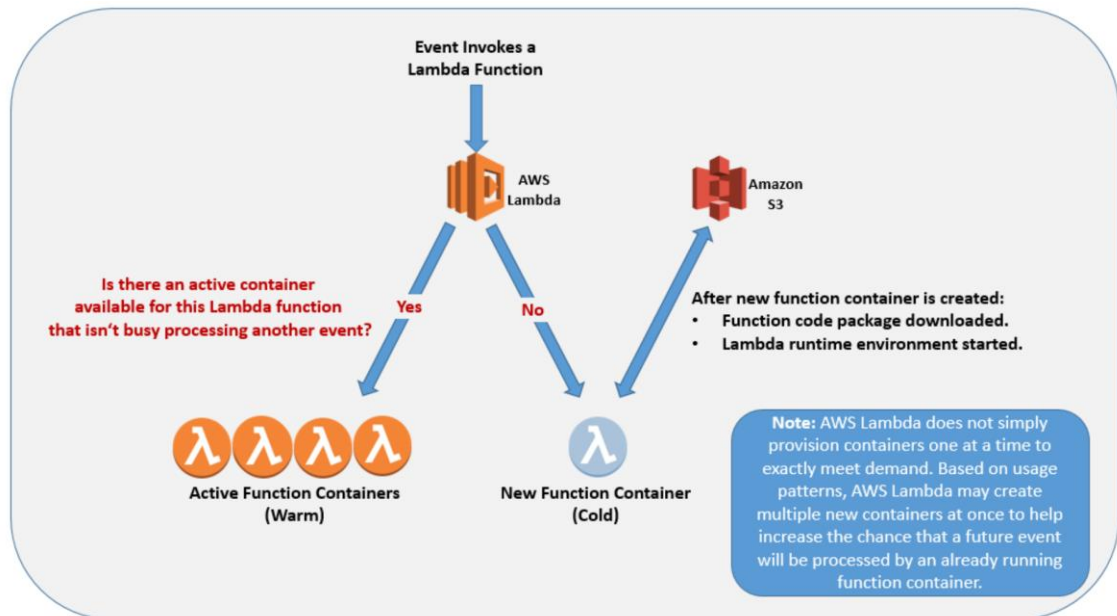


Figure 13. Invocations of warm and cold function containers [18,9]

To be extra precise, Lambda Function can be invoked in two distinct measure;

- Push Model- A case where special event takes place as the Function is invoked each and every time.
- Pull Model- Here, Lambda polls a data sources and invoke the functions with a new document at the data source, binding new source of data altogether in a particular invocation. Change in data and records on other services like DynamoDB stream is considered as one instance for a pull model. [18,10]

Likewise, synchronously and asynchronously, the Lambda Function can be also executed with selecting InvocationType parameters with RequestResponse, Event, and DryRun standards.

## Security in AWS Lambda

AWS encompasses cloud security as the highest priority. A shared responsibility between AWS and customers commonly is Lambda security.. Figure 14 shows the shared responsibility model for AWS Lambda where operating system, network configuration, underlying infrastructure, and application platforms are handled by AWS and security of their code, storage, and accessibility of sensitive data and IAM within the lambda functions are taken care by customers themselves.

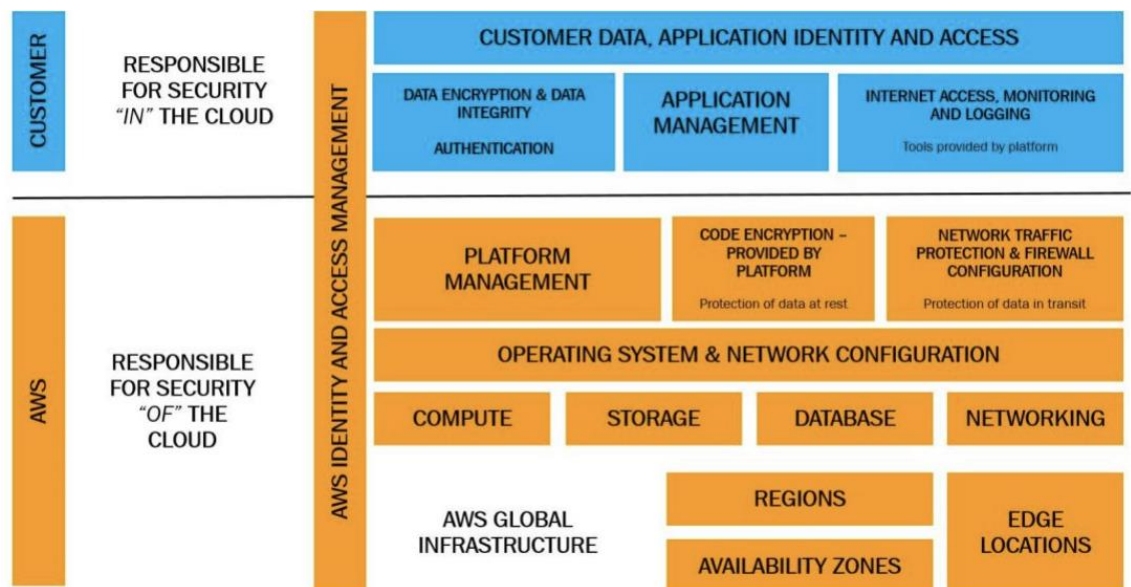


Figure 14. Shared Responsibility for AWS Lambda [19,4]

Besides, considering the following practices in a serverless environment, security in Lambda will be further strong:

- Least privilege to IAM roles- About having access to other AWS services, Lambda functions have an execution role or IAM role related to it. Developers must apply less privilege to the IAM role, meaning allowing the function to access exactly the

resources and services it needs, nothing more. It can be started giving absolute minimum privilege and later enable further permission if needed.[20,1]

- Write access to AWS CloudWatch Logs- As the Lambda function is automatically integrated with CloudWatch Logs where each event occurred can be monitor and analyze from the log stream. Providing appropriate access to Lambda logs could result in debugging and troubleshooting for the case of failure in a function. CloudWatch Logs invokes the function asynchronously with an event that contains log data. Most of the time action to create log group, log stream and put events are written in the basic execution role for Lambda as written in the below code.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

Listing 2. Role to create LogGroup [20,1]

- Avoid the same IAM role among Lambda function- Creating different IAM role for different functions as each function is assigned to achieve distinct results with a unique set of permissions is considered as a best principle for security. Ensure that Lambda functions do not share the same IAM execution role in order to promote the principle of least privilege (POLP) by granting individual function, minimum access required to handle its task. All in all, select a dedicated IAM role per Lambda function.

Similarly, reviewing API gateway security groups related to Lambda functions, granting a minimal privilege, acknowledging VPC-enabled Lambda functions and VPC endpoints will indeed level up the security in Lambda functions. [20,1]

## 4.2 Amazon API Gateway

An AWS service for creating, publishing, maintaining, monitoring, securing REST, HTTP, and web socket APIs at any scale where API developers build APIs for access different AWS services as well to other web services along with data stored in the AWS cloud. Moreover, RESTfull APIs that are HTTP-based, permit stateless client-server communication, created by API Gateway. One have to just pay for the received API calls and the volume of outgoing data that is transferred by AWS. (17) The practice of Lambda function as backend entirely possible with API methods, created with API gateway. Amazon API Gateway acts as a push invocation model with event invocation types, meaning API Gateway appears as a simple proxy as well as AWS service proxy to a Lambda function. Some use cases, web service backends such as web application, mobile app, micro-service architect, and legacy service integration are some of them. [18,11]

## 4.3 Amazon S3

An object storage service provided by AWS over the internet with an offer of industry-leading scalability, data availability, performance, and security. Working with AWS S3, users can upload unlimited files on it despite one single file that must be less than 5TB and while AWS assure 99% of availability and 99.9999999% of durable of S3. Whenever an object is created or deleted, the event notifications system permits to address events to SNS, SQS, or Lambda. To place it in detail, S3 implements the logic of buckets and objects that point out AWS to be more competitive on the market. The sequence of data, metadata, and keys are objects, and location or containers for Objects are commonly referred to as Buckets. The character of easy-to-use comprehensive the Amazon S3 to manage and configure organizational data, information. Universally, customers of several capacities readily can store and insulate their data for diverse objective as for websites, mobile applications, IoT devices, and many more. In addition, backup and disaster recovery and archive of data can be done by S3.



#### 4.4 Amazon DynamoDB

A fully managed NoSQL database service with key-value and document data structures for all applications that require consistent and single-digit millisecond latency. It is considered as a highly scalable that can handle millions of requests per second. High availability, automatic and infinite read-write I/O, instant back-up endured by DynamoDB since is serverless. Likewise, other databases, DynamoDB stores the data in tables. The core concepts of DynamoDB are tables, items and attributes where attributes are the collection of data, items are the collection of attributes and table, a location to store collection of attributes. Here, every item is defined uniquely. [17,1]

Whereas, in terms of integrating with AWS Lambda, it does invoke with the Pull model and Request/response type. With DynamoDB streams, Lambda can be triggered to perform an extra task, every time the table gets updated and Lambda poll a DynamoDB stream multiple times per second. Moreover, it can be integrated with many other AWS services, granting automatic reoccurring tasks and building applications that conclude DynamoDB as a great choice for serverless development on AWS.

#### 4.5 Amazon SNS

Amazon Simple Notification Service (Amazon SNS) is a fully managed, highly available, secure, and durable messaging service that facilitates to decouple microservices, distributed systems, and serverless applications. Moreover, it is considered as web service that delivers the messages to the subscribed endpoints or clients.

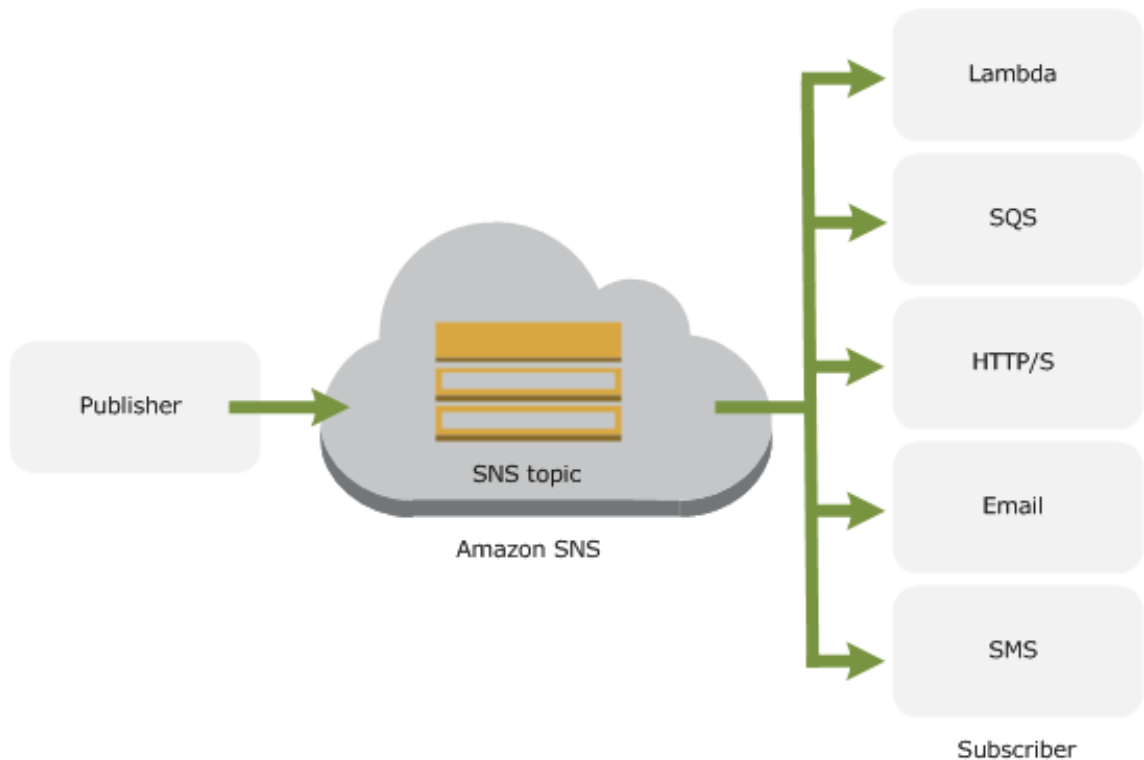


Figure 15. Amazon SNS [23,1]

Figure 15 depicts the accurate version of the process, which occurs all along with the SNS. Basically, it's communication among two parties that is publishers and subscribers. Notifications can be sent in between different services, applications, devices, and platforms through various transport protocols. The interaction among publishers and subscribers stands as asynchronously. When subscribers are subscribed to the same topic, publishers broadcast the message or notification over a reliable and portable means such as Amazon SQS, HTTPs, email, Lambda that help to receive the particular information in a second. It is indeed fascinating to observe, how the Amazon SNS filter out the notification that meant to be delivered to a specific group of consumers. A frequent and efficient measure to deliver the message makes Amazon SNS a vital part of server's development. [23,1]

## 4.6 Amazon SQS

Amazon Simple Queue Service (SQS) that fall under the messaging category as the Amazon SNS, is Amazon's distributed and fault-tolerant queuing service. It attempts to provide a secure, durable, and fully available queue to integrate and decouple shared software systems and components. It does uphold the payload of the message to 256KB and assures one-delivery of message alike to SNS. Multiple publishers and consumers are granted to interact with the same queue and built-in security that enables them to delete all the messages after the expiry period. Possessing several Pros such as secure, durable, availability, scalability, reliability, and customization made it a better Amazon service to integrate with. There are two types of queues in Amazon SQS that are standard queues which make at least one delivery and more output and FIFO queues, guarantee exactly once message delivery with strict order. Implementing the AWS key management service, the contents of the message of SQS can be made confident.

## 4.7 Amazon CloudFormation

Amazon CloudFormation assists in setting and modeling up AWS resources, established on the templates designed by the user. Normal EC2 instances or even multi-tier multi-regional applications can get maintained by the user. Getting hands-on CloudFormation is straight forward, creating a template that mentions prototype of configuring other AWS resources where template obliges as a file. Templates, stacks, and change sets are CloudFormation's concept. Here template is a JASON or YAML text file format that can be saved as in extension of .json, .yaml, .template or even .txt. [24,1] Using AWS CloudFormation bring a reliable, reproducible, and versionable deployment mechanism to deploy Lambda functions. Lambda can be specified as a custom command and provide data back to stack creation, as a factor of deploying AWS CloudFormation stacks.

## How Does AWS CloudFormation Work?

As explained previously, the stack is nothing but a single unit of resources managed in AWS services while using CloudFormation. Every stack is defined by the template of CloudFormation. It does make a call to elemental resources after creating stacks for configuration. These calls made is considered as a template. Fundamentally, it works in consecutive order- Design template on any pattern(JSON or YAML), store it locally or in S3 bucket with the appropriate file extension as .json or .yaml and lastly defining the path of template or local computer or an S3 URL create CloudFormation stack where stack can be designed using a console, API or AWS CLI. In any case, failure of the stack is managed by CloudFormation by erasing it. The code mentioned is a sample template design on YAML format which can be either written on AWS CloudFormation Designer or in a normal text editor.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: A simple EC2 instance
Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-0ff8a91507f77f867
      InstanceType: t1.micro
```

### Listing 3. CloudFormation Template

CloudFormation template can mention the EC2 instance with its characters as explained above. Besides, figure 23 (below) encapsulates the steps for creating stack for making possible to work on AWS CloudFormation.

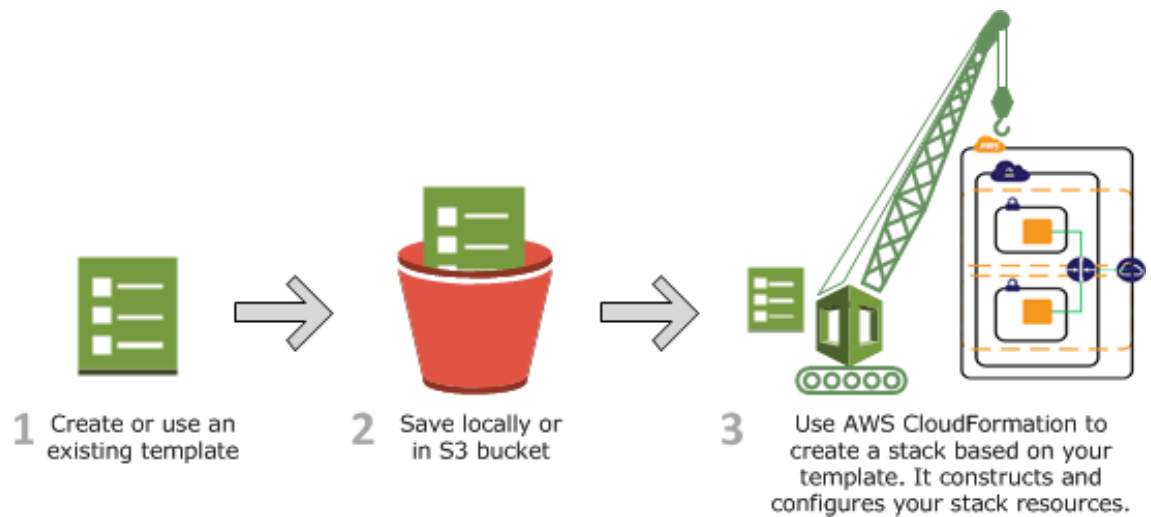


Figure 16. AWS CloudFormation Workflow [24,1]

To conclude, AWS provides many other services with the diversify principles for building cloud infrastructures virtually. As in most of the services, they are fully managed along with the gigantic range of tools for solution architect and developers. Increment in business development and growth could be grabbed in a short period with high availability, auto scalability, and security. Implementing these resources to as well as possible, a very cost-effective serverless architecture can be built.

## 5 Project on AWS: Practical Version

The following picture outlines the short project carried out to build a serverless application using Lambda Function and other AWS services. It is a serverless catalog management system that will handle the catalog, add the books with a rating, and send a notification in the form of an email to the owner whenever the books are out of stock. In the beginning, the owner uploads file to S3 with the allied information of books. Uploaded files will trigger a lambda function that will parse the files and create new items in the database of the store.

Next, adopting API Gateway will expose REST-based HTTP API that enables other systems and applications to update piles of the book along with performing other administrator operations implementing HTTP-based calls. Finally, the owner receives an email in case books run out of stock through SNS service.

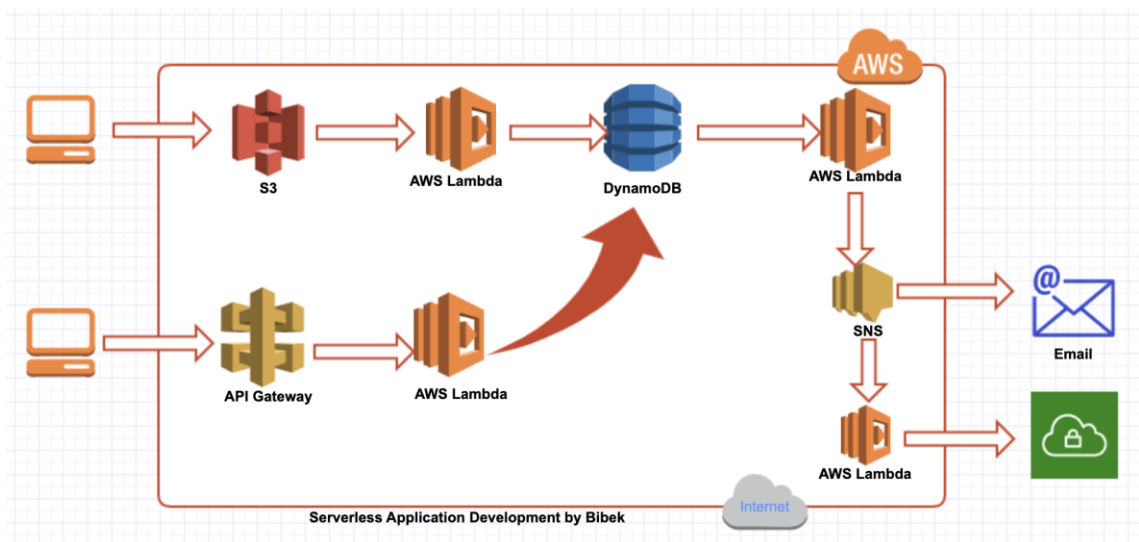


Figure 17. Diagram for Serverless Application (Personal Gliffy Account)

Figure 17 was created in Gliffy within the 14 days trial period. The Gliffy is the online platform for diagramming that works for different ideas with an easy-to-use feature.

### First Section:

In the very beginning of creating a serverless application, this is the spot where file uploaded by users to S3 will be analyzed by Lambda function as it's been configured to do so and subsequently, create a list of book item inside a DynamoDB where we need to configure S3 to make a call to Lambda function. As well, permission is required for the function to create a list of items inside DynamoDB along with reading files in S3. In favor to build catalog management, files including the Name of books, Writer, and Rating has been created in notepad that will be later uploaded to the S3 bucket. All in all, the file uploaded will be read by Lambda and form the table on the database.

To continue, let's start with creating Lambda function from scratch by filling up the name, runtime, and specific IAM role where runtime Python 3.6 will be adopted for this practice. One should be always precise while creating the role and attaching the policies within it. As Lambda function will have to access S3 and update DynamoDB, policies full access to S3 bucket and database along with the basic lambda execution should be selected. The sample JSON format written in one of the policies is attached below,

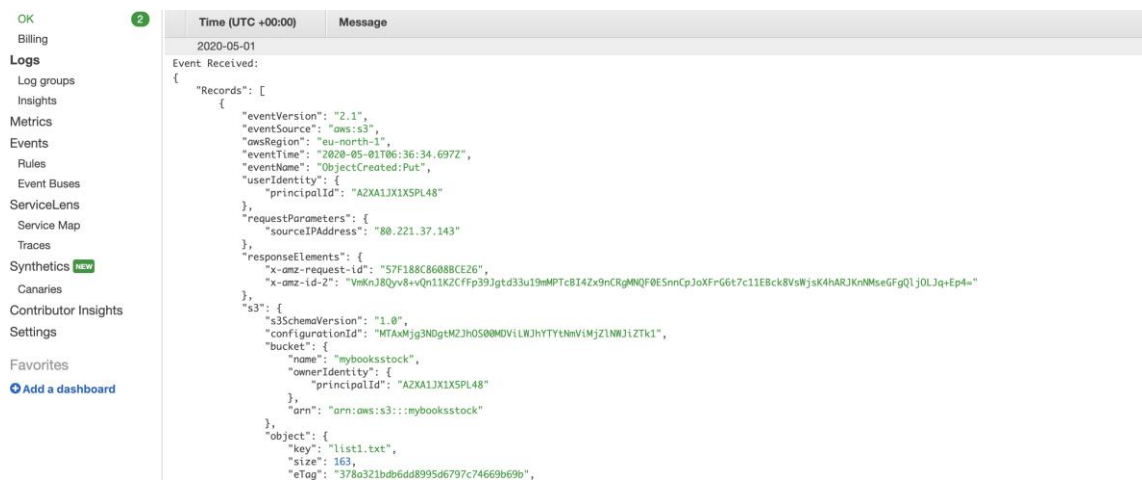
```
{
  "Version": "2012-10-17"
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem"
      "Resource": "arn:aws:dynamodb:eu:north-1:230211334822:table/catalog"
    }
  ]
}
```

Listing 4. Write access policy for DynamoDB

This specific policy will allow the lambda function to add new items to the database with the name 'catalog' as specified in the resource.

Before writing a complete Lambda Function, simple code to find out the required data passed through Lambda is written and checked to see if the function works as we wanted. Further, file to S3 bucket is uploaded granting permission 'all object create event' under the event section. As the file gets upload, a basic function created before will trigger the and logs file can be obtained from CloudWatch that will be later used while

configuring test events. Following events can be obtained from the CloudWatch Logs after triggering the following function:



```

OK
Billing
Logs
Log groups
Insights
Metrics
Events
Rules
Event Buses
ServiceLens
Service Map
Traces
Synthetics
Canaries
Contributor Insights
Settings
Favorites
Add a dashboard

Time (UTC +00:00)  Message
2020-05-01
Event Received:
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "eu-north-1",
      "eventTime": "2020-05-01T06:36:34.697Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AZKAL1X1XSPL48"
      },
      "requestParameters": {
        "sourceIPAddress": "88.221.137.143"
      },
      "responseElements": {
        "x-amz-request-id": "57F188C86088CE26",
        "x-amz-id-2": "VmkNj8Qyv8-vQn11K2Cffp39Jgtd33u19mMPTcB14Zx9nCRgMNF0ESnnCpJoxFrG6t7c11EBck8V8jsk4HAR3KnNseGfgQ1j0LJq+Ep4="
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "MTAxMjg3NDgtMzJhOS00MDVlLWJhYTYyNmVlMjZlNWJlZTk1",
        "bucket": {
          "name": "mybooksstock",
          "ownerIdentity": {
            "principalId": "AZKAL1X1XSPL48"
          },
          "arn": "arn:aws:s3::mybooksstock"
        },
        "object": {
          "key": "list1.txt",
          "size": 163,
          "eTag": "378a321bd6d8895d6797c74669b69b"
        }
      }
    }
  ]
}

```

Figure 18. Logs After triggering Lambda [Personal AWS account]

From picture 18, logs bucket name, item name and other specific details can be found to write the code that will read that file, parse it, and create new items.

Now, we can develop our code to make it further advanced to handle the events. In python, boto3 is the SDK that assists the developer to configure, manage, and create the AWS services. Here boto3 along with CSV and IO related libraries are imported. Usually, the first line of code creates a connection to the services we are going to use which is S3 and DynamoDB. By initializing the client outside of the lambda code, we allow AWS Lambda to use the existing connection again for the container to be a durable lifetime. As we now know how data is organized inside the event parameter, its easier to name the key, object in the code. The code written below will clarify the circumstances. The mentioned code is pretty straight, urllib.parse is used to transform key that can be implemented with AWS SDK. After knowing key and bucket name, S3 client can be created as 'response = s3.get\_object(Bucket=bucket, Key=key)' to retrieve the file contents. Using memory buffer and CSV library, we can read that data and transverse every role and create a new item in that database that role data as shown in code. The quantity, for now, is selected as 0. It will be further changed next phase. The code involved is written below to have a better understanding of the scenario.



```

import json
import boto3
import urllib.parse
import csv
from io import StringIO
s3 = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')

def lambda_handler(event, context):
    print("Event Received: " + json.dumps(event))

    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')

    response = s3.get_object(Bucket=bucket, Key=key)
    text = response['Body'].read().decode('utf-8')

    print("Text in file: " + text)

    buff = StringIO(text)
    reader = csv.DictReader(buff)

    table = dynamodb.Table('catalog1')

    for row in reader:
        table.put_item(
            Item={
                'Name of Book': row['Name'],
                'Rating': row['Rating'],
                'Writer': row['Writer'],
                'quantity': 0
            })

```

#### Listing 5. Lambda Function to Create Table in DynamoDB

The function written with the above code is deployed with creating a test-event that is the same as shown in figure 10. Once the code is well tested, the execution results can be observed in figure 19:

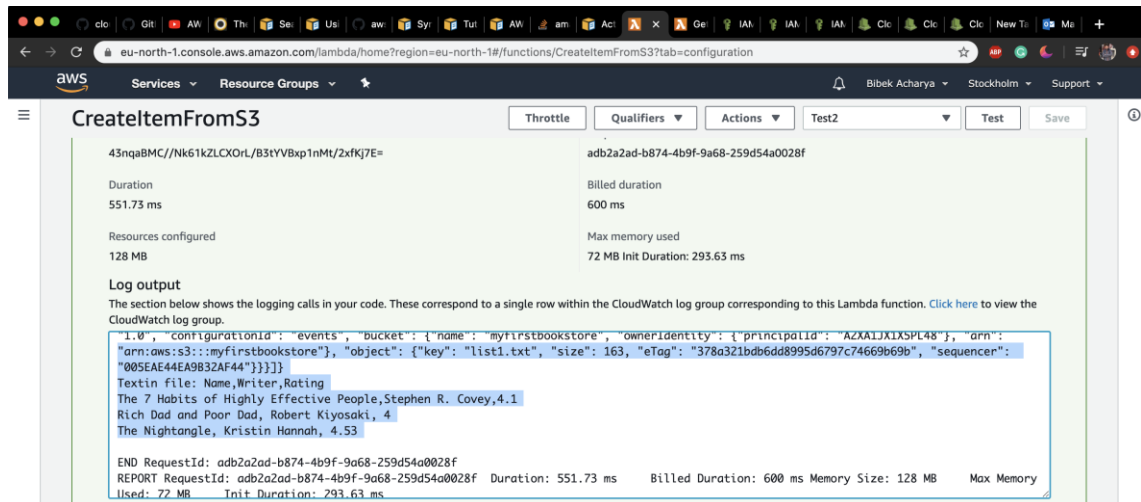


Figure 19. Result after Testing Lambda Function

After the code is well tested, we can even add other files on the S3 bucket and outcomes can be observed in the DynamoDB table and figure 20 shows the result of created table. It is quite fascinating to see that the table gets updated by the Lambda function which is one of the unique features in the serverless application. Often, it takes a while, and re-freshing the table can result in updated items.

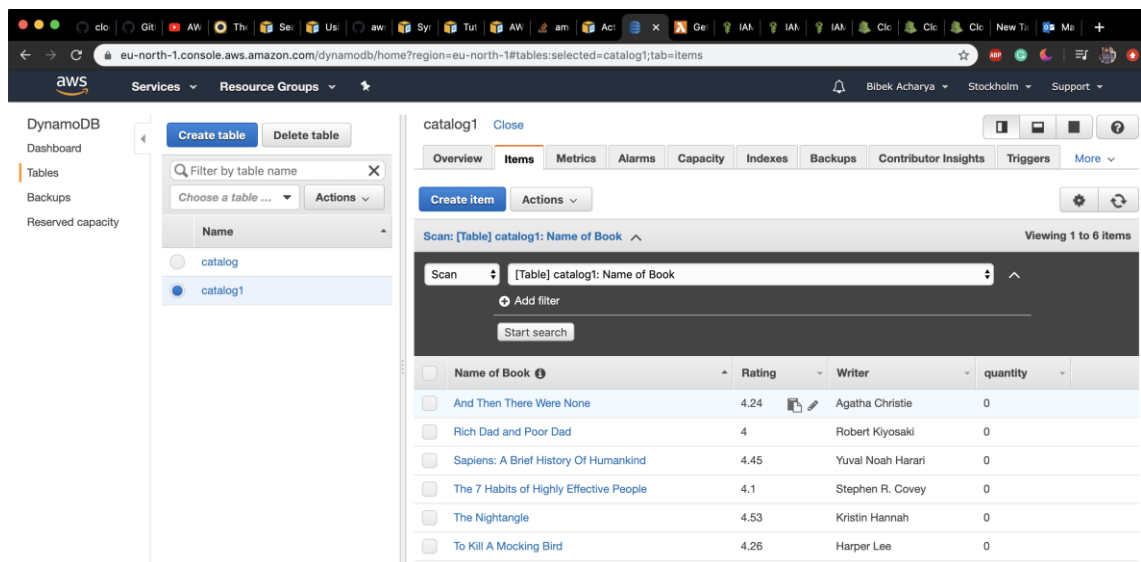


Figure 20. Updated Items Table after deploying the Lambda Function [Own AWS Account]

## Second Section:

In this section, API Gateway will be used to manage our catalog management. By creating a few endpoints that trigger the written code and update the catalog file in the database. API Gateway now will be implemented to call Lambda using specific policies. The policies attached to the role for a function to trigger API Gateway and make a change in the database will be API GatewayFullAccess, DynamoDB FullAccess, API Gateway InvokeFullAccess, and LambdaBasicExecution. It is decisive to select correct policies to mitigate the 'Access denied error' while deploying Lambda Function. For this part of the project new Function is created. We will import all the necessary libraries at the beginning of the Lambda function, proceeding with creating a client to connect DynamoDB and 'table.scan' is used to get all the items from the table. API Gateway calls the Lambda synchronously, which expects a result and it can be obtained by returning to status code and json.dumps as a response body. Further, the number of available items will be returned as a decimal value, that need to be converted to another number before serializing it to a JSON file, for which DecimalEncoder is created to return all the decimal values as float values. Now the following Python code can be tested to check its result:

```
import json
import boto3
import decimal

dynamodb = boto3.resource('dynamodb')

class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            return float(o)
        return super(DecimalEncoder, self).default(o)

def lambda_handler(event, context):
    table = dynamodb.Table('catalog')
    response = table.scan();

    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps(response, cls=DecimalEncoder),
        'headers': {
            'Content-Type': 'applications/json'
        }
    }
```

Listing 6. Lambda Function Code for Access of API Gateway

Thus, the above code on list 5 depicts the data stored in DynamoDB which now can be access through API Gateway by GET method. New API is selected with a regional

Endpoint and new methods and resources can be generated. Selecting the integration type as Lambda Function and Lambda proxy integration, function just created with the above code is saved in our new API Gateway. Clicking the test button, the status of API can be checked, and if it has been integrated as we wanted and the following output is achieved as shown in figure 21.

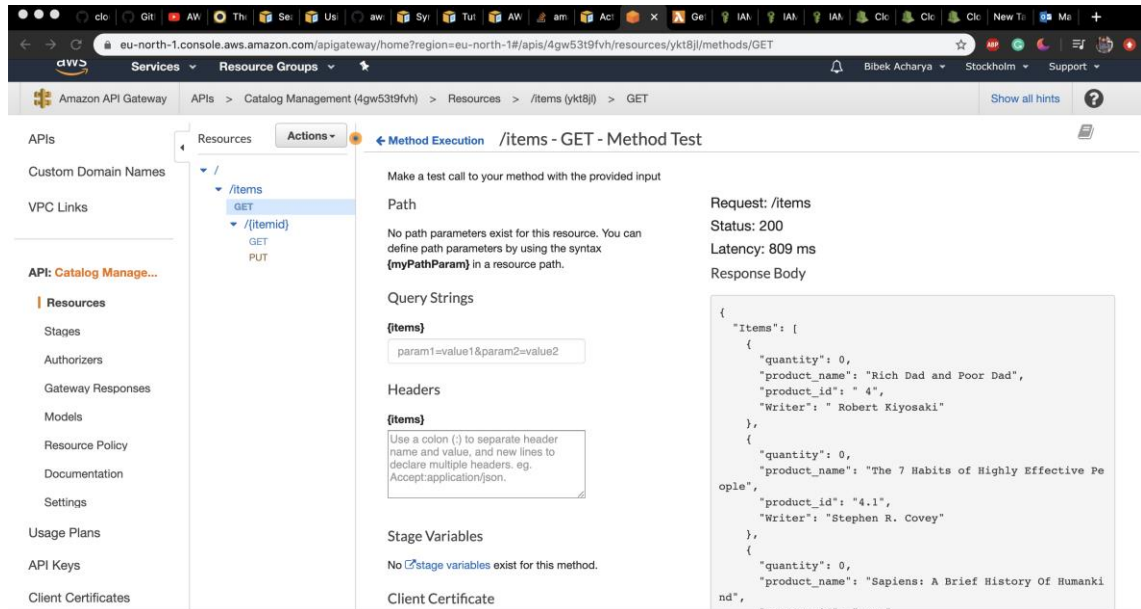


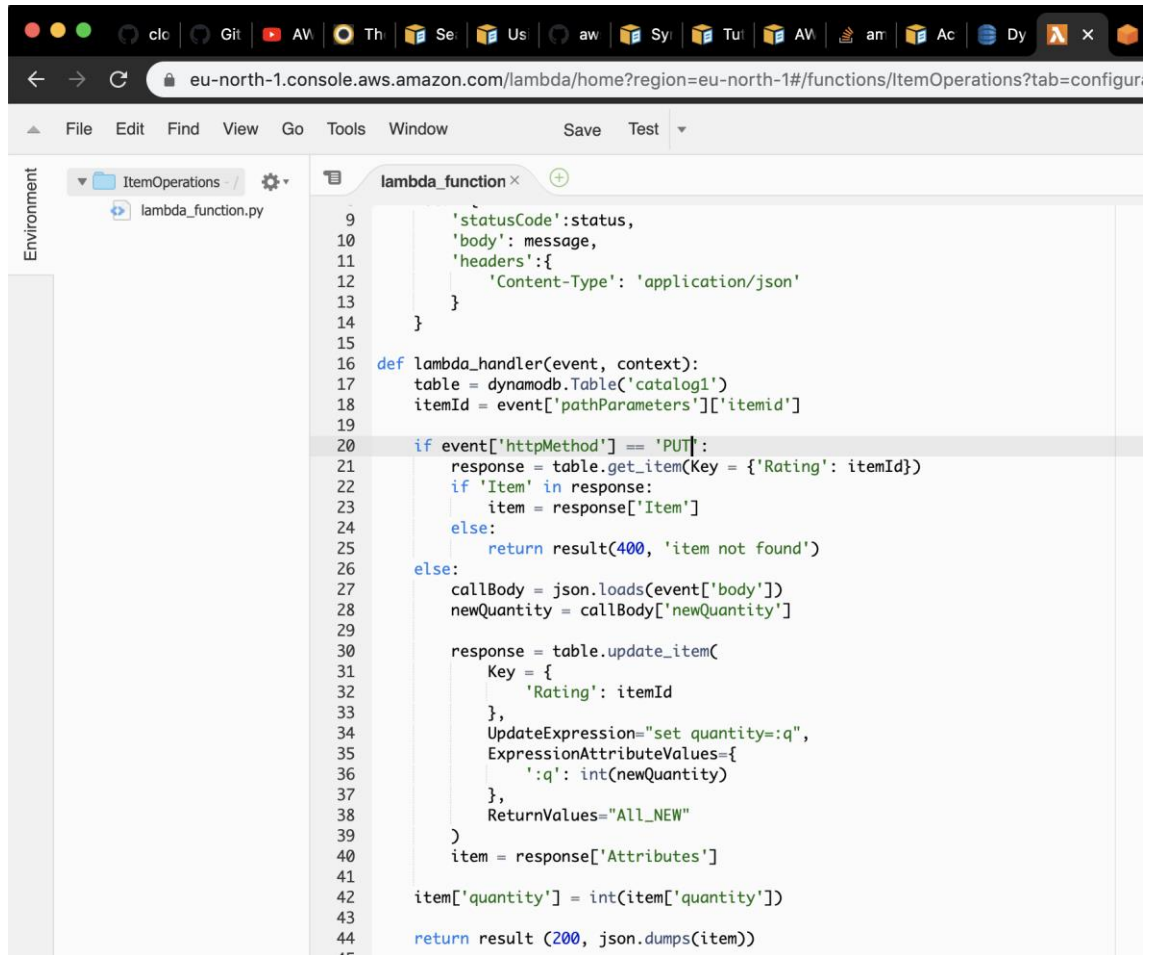
Figure 21. Items Received by API Gateway with Integrating Lambda [Own AWS account]

Similarly, the PUT method can be added to make a change in data. Now, the quantity that was zero in the first section will be altered here with the assist of PUT method. We have to implement the PUT method in Lambda function as well that can be simply done with the following code while configuring events:

```
{
  "resource": "/items/{itemid}",
  "path": "items/",
  "httpMethod": "PUT",
  "headers": null,
  "queryStringParameters": null,
  "pathParameters":
}
}
```

Listing 7. Put Method for Configuring Test Event

Finally, quantity can be changed through API Gateway with the PUT method. By mentioning { Itemid } that is the Rating of books, the quantity of the books can be made alter. We just need to update Request Body as { "newQuantity": "any number" }. As a result, significant changes in quantity can be experienced and code involved for it attached further in picture 22.



```

9         'statusCode': status,
10        'body': message,
11        'headers': {
12            'Content-Type': 'application/json'
13        }
14    }
15
16    def lambda_handler(event, context):
17        table = dynamodb.Table('catalog1')
18        itemId = event['pathParameters']['itemid']
19
20        if event['httpMethod'] == 'PUT':
21            response = table.get_item(Key = {'Rating': itemId})
22            if 'Item' in response:
23                item = response['Item']
24            else:
25                return result(400, 'item not found')
26        else:
27            callBody = json.loads(event['body'])
28            newQuantity = callBody['newQuantity']
29
30            response = table.update_item(
31                Key = {
32                    'Rating': itemId
33                },
34                UpdateExpression="set quantity=:q",
35                ExpressionAttributeValues={
36                    ':q': int(newQuantity)
37                },
38                ReturnValues="ALL_NEW"
39            )
40            item = response['Attributes']
41
42            item['quantity'] = int(item['quantity'])
43
44            return result(200, json.dumps(item))
45

```

Figure 22. Lambda Function for PUT Method of API Gateway

After deploying the code, change in quantity of books can be observed in DynamoDB table.

## Final Section:

Ultimately, we will implement the notification system with SNS service with the help of DynamoDB stream and triggering with a lambda function. Whenever the quantity of the books lowers the threshold, the provided email address will get an Email notification and later again trigger the lambda function to submit a new order. To begin with, the Dynamo stream should be enabled that will lead to figuring out the change in the item table. Latest stream ARN in the form of 'arn:aws:dynamodb:eu-north-1:230211334822:table/catalog1/stream/2020-05-03T10:06:06.661' is obtained which can be implanted create a trigger in our function. Equivalent to the previous section, we ought to have a separate IAM role with specific policies that can be a witness in figure 23,

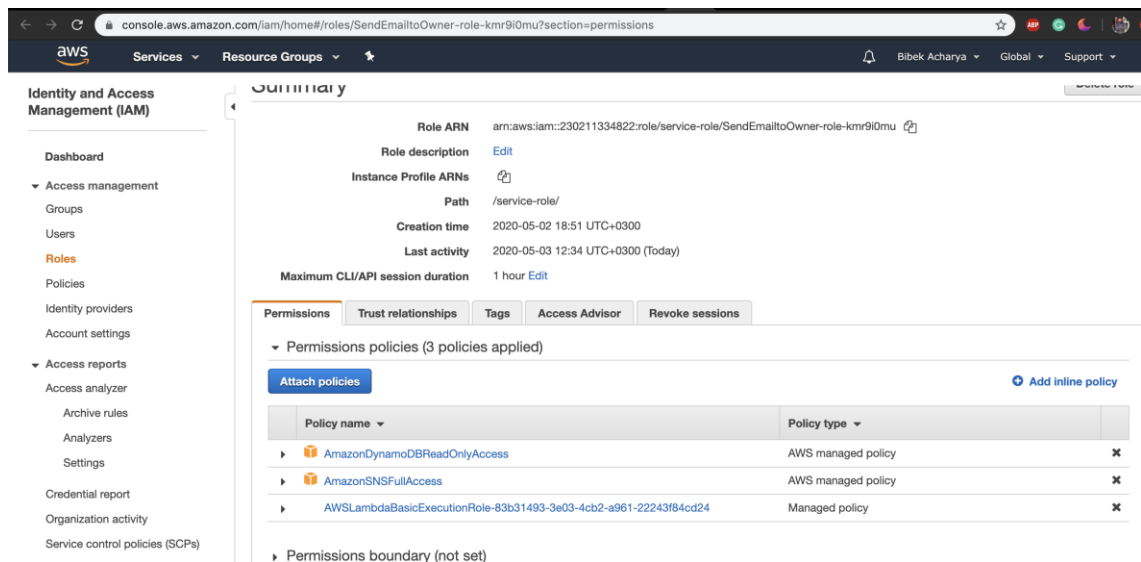


Figure 23. IAM Role with Attached Policies

Once IAM is ready, we can proceed to write a Lambda function again and attached the DynamoDB as the trigger selecting from the left corner of the configuration page. Now simple function can be called that makes sure that the book items will not be out of stock. First, we need to iterate through records and make sure the event is modified. Next updated values must be mention as a new image and get the name of the book from the key as written in code and configuration events can be taken from the log group for testing the function. Also, SNS ARN key is needed to trigger it and it does

differ from environment to environment. Here in python, to read an environment variable `os` and `os.environ` that will get the SNS ARN key and minimum quantity in which we want to trigger an alarm and send an alarm to the client using `sns.publish`. The minimum quantity for the environment variable can be selected by providing key and values below the configuration page. Now to add an SNS topic, we can create the name and get topic ARN that needs to be mention on the environment variable. Choosing protocol as email and endpoint as email-address, a subscription is created. Once the subscription is created, the owner of the email has to confirm to receive it. The code involved in Python and email received are further attached below:

```

import json
import boto3
import os
import decimal

sns = boto3.client('sns')

def lambda_handler(event, context):

    snsArn = os.environ['sns_arn']
    minQuantity = int(os.environ['min_item_quantity'])

    for record in event['Records']:
        if record['eventName'] != "MODIFY":
            continue

        newImage = record['dynamodb']['NewImage']
        if not 'quantity' in newImage:
            continue

        productId = record['dynamodb']['Keys']['product_id']['S']
        newQuantity = newImage['quantity']['N']

        print("product: " + productId + "quantity changed to: " +newQuantity)

        if(int(newQuantity) <= minQuantity):
            print("sending message to: " +snsArn)

            message = "item " + productId + "has to be added more in quantity"

            response = sns.publish(
                TargetArn=snsArn,
                Message=json.dumps({
                    'default': json.dumps(message)
                }),
                MessageStructure= 'json'
            )

```

Listing 8. Lambda Function Code to Modify Item.

## AWS Notification Message

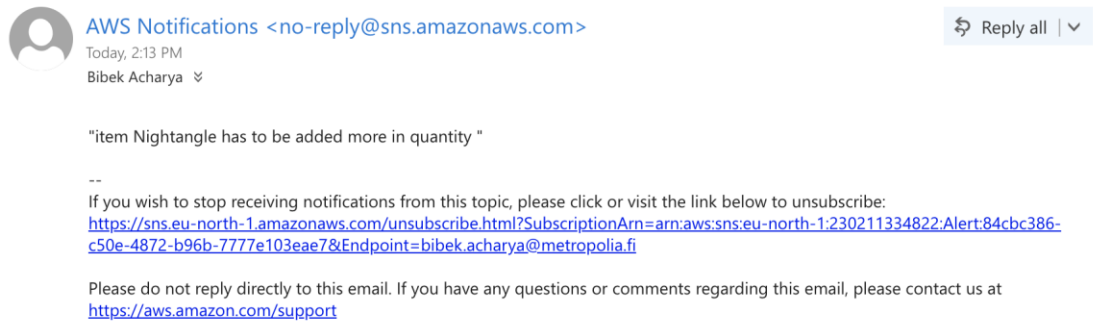


Figure 24. Alert Notification from SNS

Figure 24 shows the alert email received by the owner with the name of the specific book that need to be added to the stock. Consequently, you are answerable only for your code, all the other tasks are handled by AWS, this was the main primary objective of this project. It is quite interesting to encounter the natural action of events and functions as functions must respond to inputs, and inputs are easily modeled as an array of events.



## 6 Conclusion

Progressing for serverless with Lambda is all about understanding what serverless is and being able to illustrate what serverless does, and then fathom how to develop and create powerful applications using AWS services without ending up with large amounts of code. The prime ambition of this thesis was to establish a fully serverless application by leveraging some of Amazon's services. This study aims to explore the world of serverless beginning with the old day's big computing machines, data centers. Overcoming the drawbacks of the monolith, this final year project focuses on the virtual machines and containers with their features and working measures. Further approaching with the history of serverless, its providers and market leaders are discussed. Despite holding a few flaws, its use-cases and acceptance throughout the big and small corporations is remarkable.

To conclude, in this study AWS Lambda was thoroughly researched. The findings support a pragmatic approach to carry out the project integrating with API Gateway, S3, AWS DynamoDB, SNS, and CloudWatch. All in all, the goals for this final year project were well achieved.

## References

- 1 Muralidharan A & Thiyagarajan R. Serverless Computing- A compelling Option for Todays Digital Enterprise. [Online] Available at: [https://www.trigent.com/assets/pdf/white-paper/Trigent\\_WhitePaper\\_Serverless-Computing-A-Compelling-Option-for-Todays-Digital-Enterprise.pdf](https://www.trigent.com/assets/pdf/white-paper/Trigent_WhitePaper_Serverless-Computing-A-Compelling-Option-for-Todays-Digital-Enterprise.pdf) [Accessed 4 March 2020].
- 2 Reed J. (2018). Physical Servers vs. Virtual Machines [Online] Available at: <https://www.nakivo.com/blog/physical-servers-vs-virtual-machines-key-differences-similarities/> [Accessed 4 March 2020]
- 3 Singh J. (2018) . The Cost of Serverless.[Online] Available at <https://medium.com/faun/the-cost-of-serverless-c3fa1f8fe96> [Accessed 6 March 2020]
- 4 Corporate Shields development Team. (2019). Physical vs Virtual Servers. [online] Available at <https://www.corporateshields.com/physical-vs-virtual-servers/uncategorized/> [Accessed 6 March 2020]
- 5 RedHat. Virtualization. [online]. Available at [:https://www.redhat.com/en/topics/virtualization/what-is-virtualization](https://www.redhat.com/en/topics/virtualization/what-is-virtualization) [Accessed 6 March 2020]
- 6 LG.(2018). 14 Advantages and Disadvantages of Virtualization. [online] Available at: <https://vittana.org/14-advantages-and-disadvantages-of-virtualization> [Accessed 6 March 2020]
- 7 Patel M. (2017). Everything you Need to Know about Containers. [online] Available at:<https://medium.com/faun/everything-you-need-to-know-about-containers-7655badb4307> [Accessed 6 March 2020]
- 8 FreeCodeCamp. (2016). A Beginnere-Friendly Introduction to Containers, VMs and Docker. [online] Available at <https://www.freecodecamp.org/news/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b/> [Accessed 6 March 2020]

- 9        Gibb R. (2019). What are Containers. [Online] Available at:  
<https://blog.stackpath.com/containers/> [Accessed 8 March 2020]
- 10        Jason Ma. (2016). Serverless Architectures: The Evolution of Cloud Computing. [online] Available at:<https://www.mongodb.com/blog/post/serverless-architectures-the-evolution-of-cloud-computing> [Accessed 6 March 2020]
- 11        Duglin. (2018) Serverless Computing. [online] Available at:[https://github.com/cncf/wg-serverless/blob/master/whitepapers/serverless-overview/cncf\\_serverless\\_whitepaper\\_v1.0.pdf](https://github.com/cncf/wg-serverless/blob/master/whitepapers/serverless-overview/cncf_serverless_whitepaper_v1.0.pdf) [Accessed 8 March 2020]
- 12        Rai G, Pasricha P, Malhotra R & Pandey S. Serverless Architecture: Evolution of a New Paradigm. [online] Available at:<https://www.globallogic.com/paper/serverless-architecture-evolution-of-a-new-paradigm/5/?fbclid=IwAR3BdCqLAbTo6CNYgD4ofcdXhXaq-UDh88zAjcwB9ygRy1FuJqSPdvNeBN0#serverlessframeworks> [Accessed 8 March 2020]
- 13        Research Briefs. (2018). Why Serverless Computing is the Fastest-Growing Cloud Services Segment. [online] Available at:<https://www.cbinsights.com/research/serverless-cloud-computing/> [Accessed March 6 2020]
- 14        Castro P, Ishakian V, Muthusamy V & Slominsk V. [online] Available at:<https://arxiv.org/ftp/arxiv/papers/1906/1906.02888.pdf?fbclid=IwAR2or3SgQoNmqPbiNpBD69WBjUbX133MBXkCtNVNoZbQFKI3le5ErNgxSM> [Accessed 10 March]
- 15        Sbarski P. Serverless Architecture on AWS: With Examples using AWS Lambda. [online] Available at:[https://www.oreilly.com/library/view/serverless-architectures-on/9781617293825/kindle\\_split\\_013.html](https://www.oreilly.com/library/view/serverless-architectures-on/9781617293825/kindle_split_013.html) [Accessed 10 March]

- 16 Wayner P. (2018) Serverless in The cloud:AWS vs. Google Cloud vs. Microsoft Azure. [online] Available at: <https://www.infoworld.com/article/3265750/serverless-in-the-cloud-aws-vs-google-cloud-vs-microsoft-azure.html> [Accessed 10 March 2020]
- 17 Edelhoff R. (2019). Serverless Services on AWS. [online] Available at: <https://blogs.itemis.com/en/serverless-services-on-aws> [Accessed 11 March 2020]
- 18 Amazon Web Services, Inc. (2017). AWS Serverless Architectures with AWS Lambda. [online] Available at: <https://d1.awsstatic.com/whitepapers/serverless-architectures-with-aws-lambda.pdf> [Accessed 11 March 2020]
- 19 Amazon Web Services, Inc [online] Available at: <https://d1.awsstatic.com/whitepapers/Overview-AWS-Lambda-Security.pdf> [Accessed 11 March 2020]
- 20 Amponpun P. (2018). Steps to Secure AWS Serverless- Lambda (part1).[online] Available at: <https://medium.com/orchestrated/steps-to-secure-aws-serverless-lambda-part-1-a6e5d1b05f45> [Accessed 12 March]
- 21 Amazon Web Services, Inc [online] Available at: <https://aws.amazon.com/lambda/features/> [Accessed 10 March 2020]
- 22 Serverless. [online] Available at: <https://serverless.com/aws-lambda/> [Accessed 12 March 2020]
- 23 Amazon Web Service, Inc. What is Amazon SNS. [online] Available at: <https://docs.aws.amazon.com/sns/latest/dg/welcome.html> [Accessed 12 March 2020]
- 24 Amazon Web Service, Inc. How Does AWS CloudFormation Work? [online] Available at <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-what-is-how-does-it-work.html> [Accessed 12 March 2020]

