Understanding Serverless Computing using Amazon Web Services

[1]Shrey Kandpal

[1]Student of B.Tech (CSE)
[1]Department of Computer Science and Engineering
[1]Uttarachal University, Dehradun, India
[2]Dr. Sumit Chaudhary,
[2]Associate Professor,
[2]Uttaranchal Institute of Technology,
[2]Uttaranchal University, Dehradun

Abstract : Serverless Computing can be simply stated as development of WebApplication without worrying about the web server. Here developer working on a project need not to worry about the hardware and can simply focus on their codes, as AWS itself takes complete care of the hardware i.e Server. Server is managed by AWS officials and they provide developers with failure free IDE (Integrated Development Environment).
We will be making a simple static website which will trigger GET request. A HTML webpage is designed and been uploaded to Amazon S3 and is set to be a static website with one inex.html page and one error.html page. A click button is present inside a static website on clicking that button it will trigger a get request to the API Gateway and then that is going to trigger a lambda function which will simply print out a name (i.e in place of "Hello Everyone" I will get "Hello Shrey Kandpal") amd then that will be passed back to API Gateway and then to my S3 bucket from there it will be returned to the user.
In order to configure whole setup I first uploaded the html pages to the S3 bucket and then set the bucket permission to be static website, the html pages code is given in the code section below. Then I went to AWS lambda and created function with the name my-lambda-function where I choose runtime environment to be Python 3.6, after that created a new role with a name "myLambdaExecution" and provided it with a permission to access micro-services. I then entered my python code in IDE (Integrated Development Environment) which is stated below in the "Code Used" section. Then i created a trigger for API Gateway and added it, after which I saved the Lambda function, which further provides API endpoint. Then I configured a method inside API Gateway. I then created a new method with GET request and then set its integration type to Lambda Function and allowed the use of Lambda Proxy Integration and finally stated the name of my Lambda Function after which I deployed the API Gateway.

***IndexTerms* – Serverless Computing, AWS, Amazon Web Services, AWS Lambda, AWS S3.**

I. INTRODUCTION

Serverless Computing can be simply stated as development of WebApplication without worrying about the web server. Here developer working on a project need not to worry about the hardware and can simply focus on their codes, as AWS itself takes complete care of the hardware i.e Server. Server is managed by AWS officials and they provide developers with failure free IDE(Integrated Development Environment).

In 2006 Amazon launched EC2 and suddenly developers can provision machine with API calls from Command Line or from Web Browser. AWS is basically infrastructure as a code, as developers can provision a virtual machine anywhere in the world and have it doing whatever they want which was the birth of Infrastructure as a Service.
Then came Platform as a Service i.e. Elastic Beanstalk (initially released in 2011) where developers can just upload their code and Amazon would take a look at that code and provision the underlined resources for the developer, but still there is a need to manage OS because Amazon will not do that for the developer.
Then came containerization like Docker (initially released in 2013), they are isolated and lightweight but developers are still needed to deploy them in server and in order to keep the container running they are needed to scale them in response to load.
Then Lambda was released in reinvent 2015 and suddenly there is no need for developers to worry about managing infrastructure as a service, managing platform as a service or managing containerization as Amazon would do it all for them, all that developers need to worry about was there code. So basically AWS Lambda encapsulates developers Data Centers, Hardware's, Assembly Code/Protocols, High Level Languages, Operating Systems, Application Layer/ AWS API's.

II. CLOUD COMPUTING USING AWS

Cloud Computing can be explained as shared pool of services and configurable computer system and resources that can be quickly provisioned with very less management efforts using internet. It complete relies on sharing of resources to achieve coherence and economies of scale, similar to a public utility. Since the beginning, Amazon Web Services has made its place at the top in the field of Cloud Computing. Being so popular they launch new services nearly every week to ensure there is no competition left for them in this field.
Amazon Web Services is a subsidiary of Amazon.com which is a secure cloud services platform, offering compute power, database storage, content delivery to its customer's, who rely on AWS to launch their sophisticated applications with increased flexibility, scalability and reliability. In 2018, AWS comprised of 95+ services spanning a wide range including computing, storage, networking, database, analytics, application services, deployment, management, mobile, developer tools, and tools for the Internet of Things

Since the beginning AWS has highlighted much different architecture:

- Data Centers
- IAAS (Infrastructure as a Service)
- PAAS (Platform as a Service)
- Serverless

- Data Centers : It is the traditional way to provide infrastructure, where developers needed to send a request to the data centers providers and it would take nearly 20 to 25 days based on their location to setup datacenters. And then developers would configure it themselves, whereas in case of system failure they needed to have a backup which made the cost of managing these datacenters more challenging.

- IAAS : Infrastructure as a Service provides AWS CLI and AWS API to manage infrastructure i.e. EC2 (Elastic Cloud Computing), it was initially released by AWS in 2006 and suddenly became popular among developers. As now it would not take so much time to manage a datacenter everything can happen with few clicks. Now developers were able to deploy their virtual machines within few minutes and can delete them or scale them as they want.

- PAAS : Platform as a Service provides infrastructure based on developed applications i.e. Elastic Beanstalk. It is a service of AWS initially released in 2012 which provided more comfort to the developers. As a developer only needed to submit their code to AWS and the AWS will provision the whole server for the developer. But at this time developers were not completely free as they do need to scale there servers themselves.

- Serverless : can also be stated as FAAS (i.e. Function as a Service) provides infrastructure automatically when lambda function is executed i.e. AWS Lambda, initially released in 2014 and was known to be a game changer for developer's no worry of constructing infrastructure, no worries of scaling in or scaling out. Now developers can solely focus on their codes as AWS is going to manage the rest for them.

A. FAAS

Function as a service (FaaS) can be explained as a cloud services that enable serverless application development and management. This basically means that FaaS users are able to conduct their programming (and other tasks) without being worried about managing of their own server(s). Strings of code are triggered by events on the user end, and are simply outsourced to remote servers that are able to execute the intended functions.

Like other "as a service" models, FaaS is a method of using cloud technology to achieve higher efficiency in computer processes and workflows. It was initially introduced by hook.io in 2014, but was popularized by Amazon's AWS Lambda, as well as Microsoft Azure Functions and Google Cloud Functions and, In addition to these, there is another open-source FAAS system called OpenWhisk introduced by IBM has , the rideshare company Uber has a FaaS that runs over their private platform.

A developer may consider several use cases that are a good fit for FaaS. Some high-level cases might be:

- Scheduled tasks or jobs
- Processing a web request
- Processing queue messages
- Manually executed tasks
- Collecting, transforming, and saving collections of data
- More specific examples might be:
- Backend to Web Applications / Single Page Applications / Frontend Application
- Mobile / IoT Backends
- Data Integration / Processing services
- Chatbots
- Event or message based microservices
- Continuous content migration or incremental data imports

B. AWS Lambda

AWS Lambda can be stated as compute service that helps user's run code without managing or provisioning servers. AWS Lambda executes user's code only when needed and scales automatically, from thousands of requests per second to even few requests per day. User pay only for the compute time user consume, there is no charge when users code is not running. With AWS Lambda, user can run code for virtually any type of application or backend service - all with zero administration. AWS Lambda runs user's code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. All user need to do is supply their code in one of the languages that AWS Lambda supports (currently Node.js, Java, C#, Go and Python).

User can use AWS Lambda to run their code in response to events, such as changes to data in an Amazon S3 bucket or an Amazon DynamoDB table; to run their code in response to HTTP requests using Amazon API Gateway; or invoke their code using API calls made using AWS SDKs. With these capabilities, user can use Lambda to easily build data processing triggers for AWS services like Amazon S3 and Amazon DynamoDB, process streaming data stored in Kinesis, or create their own back end that operates at AWS scale, performance, and security.
User can also build serverless applications composed of functions that are triggered by events and automatically deploy them using AWS CodePipeline and AWS CodeBuild. For more information, see AWS Lambda Applications.

C. AWS API Gateways

API Gateway can be reffered as a main component in the cloud to connect AWS services and other private or public websites. It provides consistent restful application programming interfaces (APIs) for web applications and mobile to access AWS services. In practical terms, API Gateway lets user to create, configure, and host a restful API to enable applications to access the AWS Cloud. For example, an application can call an API in API Gateway to upload a user's daily expenditure and total income data to Amazon DynamoDB or Amazon Simple Storage Service, process the data in AWS Lambda to compute total saving, and make a record of whole month in his personal website. A user's application gains programmatic access to AWS services, or a website on the internet, through one or more APIs, which are hosted in API Gateway. The app is at the API's frontend. The integrated AWS services and websites are located at the API's backend. In API Gateway, the frontend is encapsulated by method responses and method requests, and the backend is encapsulated by integration responses and integration requests.
With Amazon API Gateway, we can build an API to provide our users with an integrated and consistent developer experience to build AWS cloud-based applications.
Together with AWS Lambda, API Gateway forms the app-facing part of the AWS serverless infrastructure. For an app to call publicly available AWS services, user can use Lambda to interact with the required services and expose the Lambda functions through API methods in API Gateway. AWS Lambda runs the code on a highly available computing infrastructure. It performs the necessary execution and administration of the computing resources. To enable the serverless applications, API Gateway supports streamlined proxy integrations with AWS Lambda and HTTP endpoints.

D. AWS DynamoDB

DynamoDB is a fully managed NoSQL database service that allows users to create database tables that can retrieve and store any amount of data. It automatically maintains performance and manages the data traffic of tables over multiple servers. It also relieves users from the burden of operating and scaling a distributed database. Hence, hardware provisioning, replication, configuration, setup, cluster scaling, software patching, etc. is managed by Amazon itself.

E. AWS S3

Amazon Simple Storage Service is storage for the Internet, in simple ways a person can think of it as Google Drive. It is designed to make web-scale computing easier for developers.
Amazon S3 has a simple web services interface that user can use to retrieve and store any amount of data, at any time, from anywhere on the web. It gives any developer access to the same reliable, highly scalable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers.
Amazon S3 is cloud storage for the internet. To upload user's data (photos, videos, documents etc.), user first create a bucket in one of the AWS Regions, and can then upload any number of objects to the bucket.
In terms of implementation, objects and buckets are resources, and Amazon S3 provides APIs for user to manage them. For example, user can create a bucket and upload objects using the Amazon S3 API. User can also use the Amazon S3 console to perform these operations. The console uses the Amazon S3 APIs to send requests to Amazon S3.

III. ILLUSTRATIONS

In this illustration we will be making a simple static website which will trigger GET request. A HTML webpage is designed and been uploaded to Amazon S3 and is set to be a static website with one inex.html page and one error.html page. A click button is present inside a static website on clicking that button it will trigger a get request to the API Gateway and then that is going to trigger a lambda function which will simply print out a name (i.e in place of "Hello Everyone" I will get "Hello Shrey Kandpal") amd then that will be passed back to API Gateway and then to my S3 bucket from there it will be returned to the user.

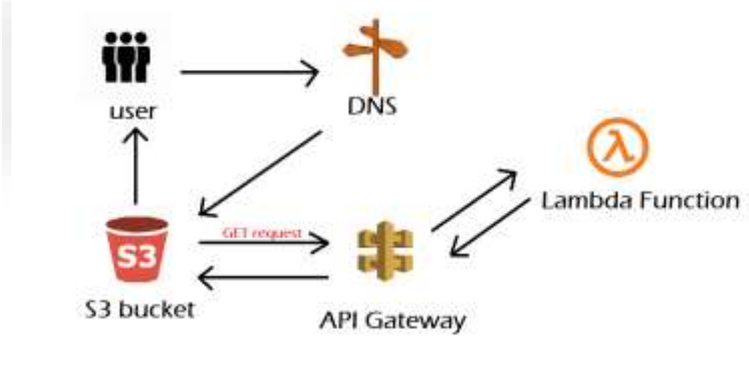

Figure 1. Architecture of Illustration

In order to configure whole setup I first uploaded the html pages to the S3 bucket and then set the bucket permission to be static website, the html pages code is given in the code section below. Then I went to AWS lambda and created function with the name my-lambda-function where I choose runtime environment to be Python 3.6, after that created a new role with a name "myLambdaExecution" and provided it with a permission to access micro-services I then entered my python code in IDE (Integrated Development Enviroment) which is stated below in the "Code Used" section. Then i created a trigger for API Gateway and added it, after which I saved the Lambda function ,which further provides API endpoint. Then I configured a method inside API Gateway. I then created a new method with GET request and then set its integration type to Lambda Function and allowed the use of Lambda Proxy Integration and finally stated the name of my Lambda Function after which I deployed the API Gateway

That's it so whenever I run my static website it will show me the result as stated in fig 4 and fig 5.

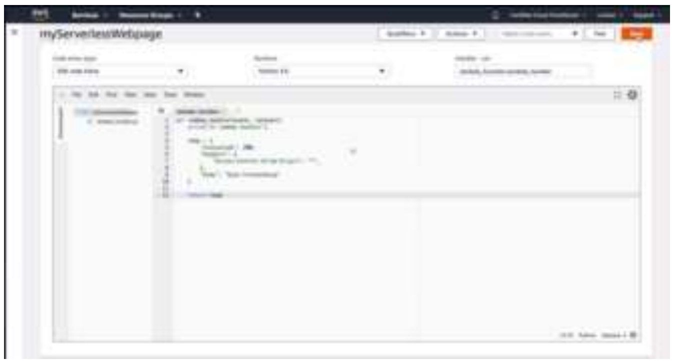
Figure 2. GET Method Configuration

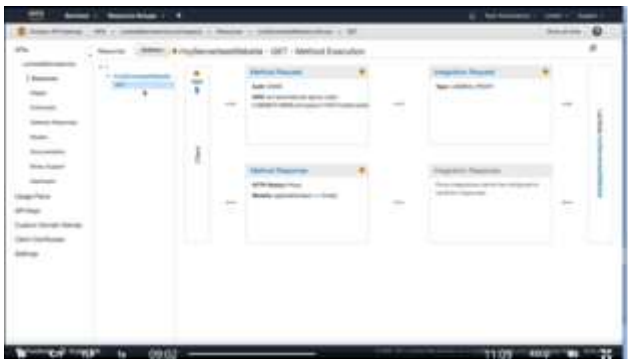
Figure 3. GET Method Configuration

IV. CODES USED

For static HTML Site in S3 :

- Index Page :

```
<html>
<head>
<script>
function myFunction()
{
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function()
{
if (this.readyState == 4 && this.status == 200)
{
document.getElementById("my-demo").innerHTML = this.responseText;
}
};
xhttp.open("GET","https://9xx82lkvp8.execute-api.us-east-
1.amazonaws.com/prod/MyServerlessWebsite",                true);
xhttp.send();
}
</script>
</head>
<body>
<div align="center">
<br>
<br>
<br>
<br>
<h1>Hello <span id="my-demo">Everyone!</span></h1>
<button onclick="myFunction()">Click me</button>
<br>
<img   src="https://s3.amazonaws.com/static-site-s3-bucket/my-
image.JPG">
</div>

</body>
</html>
```

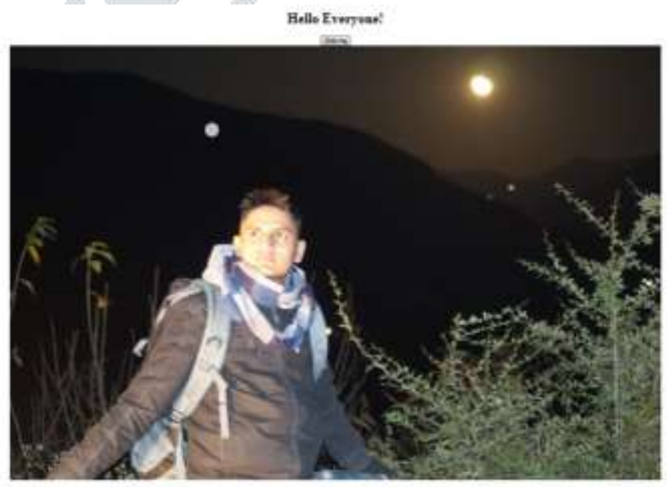

Figure 4. Before Lambda Trigger

- Error Page :

```
<html>
  <head></head>
  <body>
```

```
    <h1>There has been an error!</h1>
  </body>
</html>
```

- Lambda trigger in Python IDE :

```
def lambda_handler(event, context):
   print("In lambda handler")

   resp = {
      "statusCode": 200,
      "headers": {
         "Access-Control-Allow-Origin": "*",
      },
      "body": "Shrey Kandpal"
   }

   return resp
```

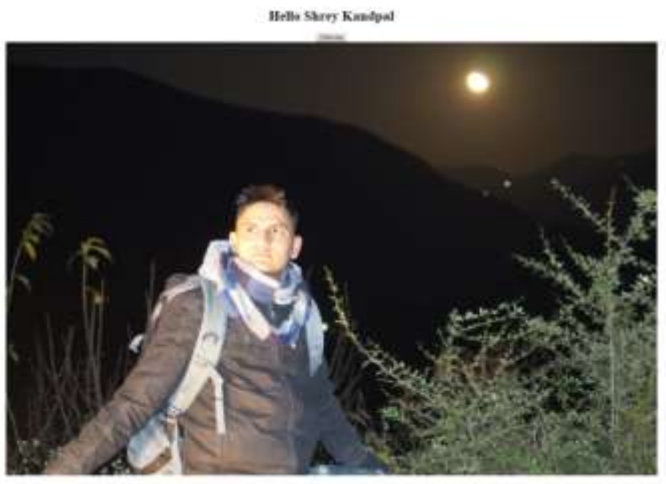
Hello Shrey Kandpal

Figure 5. After Lambda Trigger

V. CONCLUSION

Serverless computing is becoming a base requirement in the world of cloud. With the increase in the cost of maintaining servers developers are less concentrated on their code, and rather more concentrated on reducing the optimization cost of a server and in preventing it from unintentional failure. With the increase in load on the server there are increased chances of breakdown. And hence there is a need of a solution. Amazon Lammda is severless based platform which helps developers to work on their code without worrying about the server and deploy it without any second thoughts of failiure. The server is completely taken care by the team deployed by AWS. In this paper it is shown how a person can use these services in order to make a well optimized, cost efficient and failure free, web application with the use of different serverless services provided by AWS without the use traditional web servers. It is very essential for developers working on cloud to understand serverless, as it may help them save loads of time.

VI. REFERENCES

[1]Serverless Computing: Design, Implementation, and Performance by Garrett McGrath ; Paul R. Brenner.

[2]Serverless Programming (Function as a Service) by Paul Castro ; Vatche Ishakian ; Vinod Muthusamy ; Aleksander Slominski.

[3]Amazon API Gateway Doc. [online] available: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

[4]Amazon Simple Storage Service Documentation [online] available: https://docs.aws.amazon.com/s3/index.html

[5]Serverless Computation with OpenLambda Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani.

[6]Status of Serverless Computing and Function-as-a-Service(FaaS) in Industry and Research by Geoffrey C. Fox, Vatche Ishakian, Vinod Muthusamy, Aleksander Slominski (Submitted on 27 Aug 2017).

[7]Overview of Amazon Web Services [online] available: https://docs.aws.amazon.com/aws-technical-content/latest/aws-overview/introduction.html

[8]An Introduction to serverless and FAAS: https://www.phase2technology.com/blog/introduction-to-serverless-and-faas

[9]Serverless Computing: An Investigation of Factors Influencing Microservice Performance by Wes Lloyd ; Shruti Ramesh ; Swetha Chinthalapati ; Lan Ly ; Shrideep Pallickara.

[10]Cloud Computing with AWS [online] available: https://aws.amazon.com/what-is-aws/