

SUMMER INTERNSHIP REPORT

A REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE AWARD OF DEGREE OF

BACHELOR OF ENGINEERING

IN

ELECTRONICS & COMMUNICATION ENGINEERING

BY

PALOJI SAI VARMA

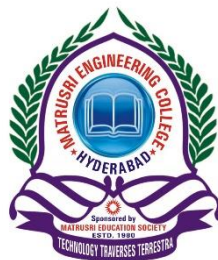
(1608-19-735-019)

Under the Supervision of

Mr. P. RAVI KUMAR REDDY

M.E, (Ph. D)

Assistant Professor



Department of Electronics and Communication Engineering

(Accredited by NAAC A+ & NBA)

MATRUSRI ENGINEERING COLLEGE

(Sponsored by Matrusri Education society, Estd1980)

(Approved by AICTE, Affiliated to Osmania University)

#16-1-486, Saidabad, Hyderabad, Telangana-500 059

www.matrusri.edu.in

2021-22



Matrusri Engineering College

(Sponsored by : MATRUSRI EDUCATION SOCIETY, Estd: 1980)

(Approved by AICTE, Affiliated to Osmania University)

16-1-486, Saidabad, Hyderabad - 500 059. Ph : 040-24072764

Email : matrusri.principal@gmail.com , www.matrusri.edu.in



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Date: 10.01.2023

Certificate

This is to certify that the “SUMMER INTERNSHIP REPORT” submitted by **Mr. Paloji Sai Varma** Roll No. **1608-19-735-019** is work done by him and submitted during 2022-23 Academic year, in partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Electronics and Communication Engineering of the Osmania University, Hyderabad.

P. Ravi Kumar Reddy
Asst. Professor
Internship Coordinator.

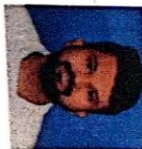
Dr. N. Srinivas Rao
Head of the Department



एन एस आई सी
N S I C

राष्ट्रीय लघु उद्योग निगम-तकनीकी सेवा केन्द्र
THE NATIONAL SMALL INDUSTRIES CORPORATION LTD.
TECHNICAL SERVICES CENTRE

(भारत सरकार का उद्यम / A Government of India Enterprises)
ई.सी.आई.एस.एल. एक्स रोड, कुशागुडा, हैदराबाद - 500062, तेलंगाना, भारत
E.C.I.L. X Road, Kushaiguda, Hyderabad - 500062, Telangana, India.



क्रमांक / S.No. 188591

दिनांक / Date: 31/05/2022

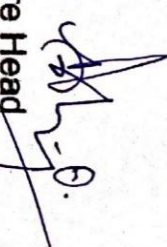
Certificate

This is to certify that Mr. / Ms. Paloji Sai Varma
son/daughter of Mr. Paloji Srinivas pursuing BE in ECE from
(College Name) Matrusri Engineering College, Saidabad
Roll No. 1608-19-735-019 has successfully completed the Internship Program
entitled/in the area of Amazon Web Services under

our guidance. It is a bonafide work carried out by her/him from 17/05/2022 to 31/05/2022
He/She has completed the assigned module as per the requirements within the time frame
During the above period, the trainee's conduct was found Good


Project Coordinator




Centre Head

यह प्रमाण पत्र धोतोलक्षण तथा हेतवे के साथ ही मान्य होगा / This Certificate shall be valid only with affixed hologram

ACKNOWLEDGEMENT

First, I would like to thank **Lokesh**, the Director of **National Small Industries Corporation Technical Service Centre (NSIC)** giving me the opportunity to do a SUMMER INTERNSHIP within the Organization.

I also would like all the people that worked along with me with their patience and openness they created an enjoyable working environment.

It is indeed with a great sense of pleasure and immense sense of gratitude that I acknowledge the help of these individuals.

I am highly indebted to Principal **Dr. D. Hanumantha Rao**, for the facilities provided to accomplish this Internship.

I would like to thank my Head of the Department **Dr. N. Srinivasa Rao**, for his constructive criticism thought-out my internship.

I would like to thank **P. Ravi Kumar Reddy**, Department Internship Coordinator for his support and advices to get and complete internship in above said Organization.

I am Extremely grateful to my department staff members and friends who helped me in successful completion of this internship.

PALOJI SAI VARMA

1608-19-735-019

ABSTRACT

Cloud Computing in recent times has unfolded as the most popular and efficient standard for managing and delivering useful services over the internet. Out of various cloud services and techniques, serverless computing represents an evolution of cloud-based programming technologies, and is an attestation to the growth and large-scale adoption of cloud concepts. In this era of increasing technological advancements large internet companies like Amazon, Netflix, and LinkedIn deploy big multistage applications in the cloud which can be developed, tested, deployed, scaled, operated and upgraded independently. However, aside from gaining agility and scalability, infrastructure costs are a major hindrance for companies following this pattern due to the increasing server loads and the need to increase server space. This is where serverless computing and services like AWS Lambda comes into picture. The paper delves into the functioning of AWS Lambda along with other existing AWS services through the development of a serverless chat application which supports scalability without the addition of new servers. The paper further explores the connection of different existing services in AWS like S3, DynamoDB, CloudWatch etc. with serverless technologies like Lambda.

Keywords: AWS, Lambda, S3, DynamoDB, Cognito, SNS, IAM, Serverless, CORS, API Gateway

INDEX

TOPIC	PAGE NO
ORGANIZATION PROFILE	7
INTERNSHIP OBJECTIVES	8
INTRODUCTION	9-12
SERVERLESS	13-16
AMAZON WEB SERVICES	17-23
LITERATURE SURVEY	24
METHODOLOGY OF THE PROJECT	25
IMPLEMENTATION OF THE PROJECT	26-39
RESULTS	40-41
CONCLUSION	42
WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES	43-44
REFERENCES	45

ORGANIZATION PROFILE

National Small Industries Corporation Limited (NSIC) is a Mini Ratna government agency established by the Ministry of Micro, Small and Medium Enterprises, Government of India in 1955. It falls under Ministry of Micro, Small & Medium Enterprises of India. NSIC is the nodal office for several schemes of Ministry of MSME such as Performance & Credit Rating, Single Point Registration, MSME Databank, National SC ST Hub, etc.

It was established in 1955 to promote and develop micro and small-scale industries and enterprises in the country. It was founded as a Government of India agency later made into a fully owned government corporation.

The National Small Industries Corporation Ltd. (NSIC), is an ISO 9001-2015 certified Government of India Enterprise under Ministry of Micro, Small and Medium Enterprises (MSME).

NSIC operates through countrywide network of offices and Technical Centre's in the Country. To manage operations in African countries, NSIC operates from its office in Johannesburg, South Africa. However, From January 2018, Johannesburg office is now closed and is now closely looking after domestic MSME Units. In addition, NSIC has set up Training cum Incubation Centre & with a large professional manpower, NSIC provides a package of services as per the needs of MSME sector.

NSIC has recently partnered with Rubique.com, to facilitate lending for MSME segment. Rubique & NSIC will work together to create an interface which will ease credit facilitation for MSMEs by allowing quicker decision making and evaluation and to widen the product offerings will bring their respective bank/FI tie-ups under one umbrella for MSME.

NSIC also helps in organizing supply of raw materials like coal, iron, steel and other materials and even machines needed by small scale private industries by mediating with other government companies like Coal India Limited, Steel Authority of India Limited, Hindustan Copper Limited and many others, who produce these materials to provide same at concessional rates to SSIs. Further, it also provides assistance to small scale industries by taking orders from Government of India owned enterprises and procures these machineries from SSI units registered with them, thus providing a complete assistance right from financing, training, providing raw materials for manufacturing and marketing of finished products of small-scale industries. It also helps SSI by mediating with government owned banks to provide cheap finance and loans to budding small private industries of India.

Nowadays, it is also providing assistance by setting up incubation centers in other continents and also international technology fairs to provide aspiring entrepreneurs and emerging small enterprises a platform to develop skills, identify appropriate technology, provide hands-on experience on the working projects, manage funds through banks, and practical knowledge on how to set up an enterprise.

INTERNSHIP OBJECTIVES

- To develop skills in the application of theory to practical work situations.
- To develop skills and techniques directly applicable to their careers.
- To increase the student's sense of responsibility and good work habits.
- Exposure to real work environment, experience & gain knowledge in writing report in technical works/projects.
- To build the strength, teamwork spirit and self-confidence.
- To enhance the ability to improve creativity skills and sharing ideas.
- To build a good communication skill with group of workers and learn to learn proper behavior of corporate life in industrial sector.
- Learning good moral values such as responsibility, commitment and trustworthy during their training.

1.INTRODUCTION

Organizations such as WhatsApp, Instagram and Facebook have long been promoting conversational UI (User Interface) based applications that are based on either text or voice or both. This has recently resulted in growing interest regarding how relevant technologies could be used to improve business in wide industry domains.

The latest trend in cloud computing constructs is the use of serverless architecture [1]. This fast-developing cloud model includes the provision of a server and its managerial operations by the cloud provider itself which eliminates the need for the user to maintain, monitor and schedule servers. This greatly simplifies the methodology of deploying the code into production process. Serverless code is employed in conjunction with code deployed in generic code designs, like microservices. Majority of the serverless applications run in state-less compute blocks that are triggered by events. The pricing depends on the number of executions rather than a already purchased number of compute capacity servers.

This paper reports a month-long case study where a chat application was built based on Amazon Web Services' Lambda framework which as mentioned above, is a rapidly evolving serverless computing [2] platform. The paper also covers explorations about various services and components of Amazon Web Services and their usage [3].

1.1 Background

Serverless computing or in short serverless affects not only how applications are run but also how they are built. The serverless application programming model is based on best practices of development in a cloud computing environment. Cloud computing began with services offering to rent infrastructure instead of buying it. However, managing virtual servers still requires a lot of labor of intensive work, but that work can be automatized. Automatization was the primary driving force of DevOps culture. DevOps people write scripts to deploy, provision, scale, and monitor virtual servers and applications. Those scripts become tools like puppet and chef. Afterward platforms such as Heroku and Kubernetes emerged. DevOps experience, knowledge, and best practices become part of technology, and the latest product of that is serverless.

Scania IT is interested in serverless because this new architecture can simplify some types of systems and it can provide economic benefits by reducing operation and infrastructure costs. At the moment Scania owns and maintains 3 data centers across the world, in total there are about ~500 physical servers and ~5500 virtual servers. However, their business model is not to take care of servers, and the company's strategy is to move from infrastructure on-premise to a cloud. Serverless Computing is the ideal solution because there is no management of server hosts or server processes.

This thesis describes Serverless application programming model, in other words, how applications can be built and hosted in Serverless environment. The primary focus of this thesis is on Amazon Web Services (AWS) because Scania IT has chosen AWS as their main cloud computing platform. It is worth to mention, that thesis is not about AWS Serverless Application Model (AWS SAM) which provides a standard for expressing Serverless applications on AWS. However, section 4.2 Frameworks includes an analysis of AWS SAM.

1.2 Problem

Serverless computing is a new cloud computing execution model, and the problems of it are still in an exploration phase. Organizations are not fully aware of models, benefits and challenges building applications with Serverless architecture. That leads to economic losses when serverless is chosen and it can't meet requirements. However, more often, opportunity to reduce costs is missed due misunderstandings of serverless limitations, such as cold-start and stateless execution.

1.3 Purpose

The purpose of this thesis is to present serverless application programming model, describe the advantages and challenges from a software engineering, development productivity (both design, coding, testing, and debugging), performance, scalability and security perspectives. Presented material should help to understand when serverless architecture is the right choice and give the necessary knowledge to avoid common pitfalls when working with the serverless stack.

1.4 Goal

The goal of this project is to gather necessary knowledge which would help successfully use serverless computing. That includes:

1. Describe how serverless computing works.
2. Analyze deployment package size impact on cold-start
3. Compare serverless development frameworks.
4. Identify when serverless architecture is applicable.
5. Describe challenges and best practices of serverless application programming model.

1.4.1 Benefits

This is beneficial to everyone who uses serverless computing services because it helps to understand better what serverless is, how it works and when it is the right choice to use.

Results of this project are especially beneficial for the developers. It provides necessary information to start working with serverless stack and gives instructions on how to determine if application requirements can be fulfilled using serverless architecture, thus it reduces learning-curve and wasted efforts due to missing knowledge about serverless.

1.5 Methodology

This thesis tries to provide the necessary knowledge needed to start working with Serverless stack, therefore, applied method was chosen. Data were collected using literature study, two qualitative methods: interviews and case study and quantitative to analyze cold-start problem. The methodology of it is presented in Section 3.1.

An extensive literature study was needed to describe the Serverless Application Programming model because serverless is a new phenomenon which is driven by industry. Thus, available information is limited in order to protect trading secrets and sometimes outdated because technology evolves rapidly. Literature study includes material not only from previous es but also from whitepapers, conferences and blog posts due to freshness and completeness of information.

Multiple informal interviews were conducted with various people to learn how they understand and how they are using serverless and what are the impressions and concerns about it. Interviewees were developers, infrastructure engineers, software architects and managers at Scania IT, also solution architect from AWS.

The case study was carried out to evaluate the following serverless frameworks: Chalice, Apex, AWS SAM, Up and the Serverless Framework. To evaluate eight simple APIs with one endpoint was built in Java 8, C# (.NET Core 2.0), Go, Node.js (6.10, 8.10) and python3 languages and deployed to AWS. User experience was evaluated looking at how easy it is to set up a framework, prepare project and deploy the application. It was done using a computer with Windows 10.

Additionally, some observations are from personal experience gained during one year of developing REST API on AWS serverless stack.

1.6 Delimitations

Serverless field evolves quickly, new tools are released, and existing ones are improved constantly, therefore features of serverless frameworks described in section 4.2 Frameworks might be inaccurate at the time of reading. The same applies to AWS features and measures because AWS is constantly working to improve performance and user experience of the serverless platform and keeping details in private. Therefore, measures and descriptions how it works are only to provide an overall view about a serverless platform and give a general idea what to expect.

1.7 Outline

This report consists of three main parts. The first part is a general introduction to serverless, serverless computing and AWS serverless platform. It is presented in Chapter 2.

The second part, presented in Chapter 3, analyzes the impact of deployment package size on cold-start. That chapter describes experiments, presents results and discusses them.

Chapter 4 presents the third and the main of the project - Serverless Application Programming model. It covers topics about serverless application development, architecture, security, observability, optimizations and limitations.

Lastly, Chapter **Error! Reference source not found.** summarizes results of the project, provide some conclusions, and give suggestions for possible future work.

2. Serverless

The serverless is a new generation of platform-as-a-service which distinguishing feature is no maintenance of servers. Also, Serverless is driven by industry, AWS in particular, which is visible looking at popularity between “Serverless” and “AWS Lambda” search terms. Figure 1 depicts raise of “AWS Lambda” searches followed by “Serverless” in google.

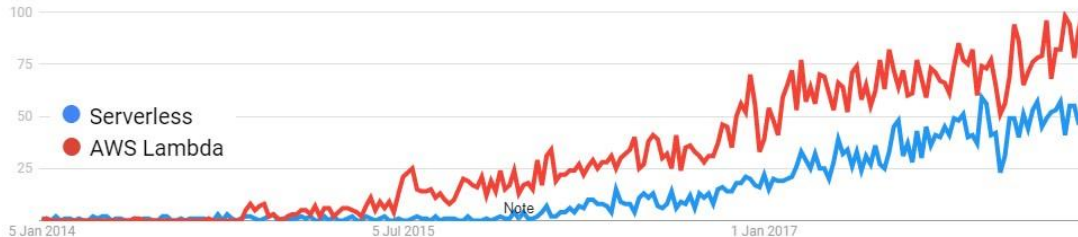


Figure 1 Popularity of Serverless and AWS Lambda search terms in google between 2014-2018-04

2.1 Definition

There is still no stable definition what serverless is, however most widespread description what serverless means is provided by Mike Roberts:

1. No management of server systems or server applications
2. Horizontal scaling is automatic, elastic, and managed by the provider.
3. Costs based on precise usage
4. Performance capabilities defined in terms other than size or count of instances
5. Implicit high availability.

Serverless can be divided into two overlapping areas: Function as a Service and Backend as a Service (managed services).

Serverless Function as a Service, also known as Serverless computing, is a service which allows running application logic on stateless compute containers that are fully managed by a third party. Serverless computing is the type of Function as a Service, which is part of event-driven computing, Figure 2 depicts the relation.

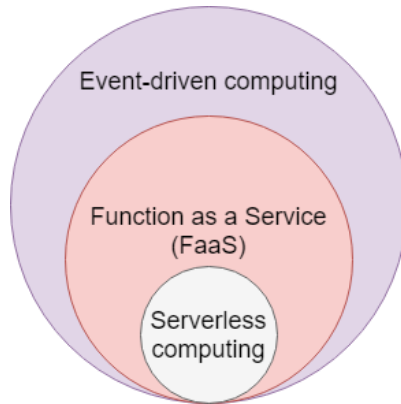


Figure 2 Relation between Serverless computing, Function as a Service and Eventdriven computing

Fintan Ryan, an industry analyst at RedMonk, described [1] serverless computing as:

- programming model to deal with event-driven architecture
- abstraction layer, which abstracts underlying infrastructure

Applications running on serverless computing most of the time relies on other provided services, which falls under Backend as a Service category. CNCF Serverless Working Group defines Backend as a Service (BaaS) as “third-party API-based services that replace core subsets of functionality in an application” [2].

To sum up, despite how it sounds servers are and continue to be necessary to run serverless services, the same as wires are needed for Wireless networks. The “less” just emphasize the idea that servers are hidden from business who pay for services and developers who uses them. Those services scale automatically, operate transparently and do not cost when idle.

2.2 Serverless Processing

This section describes how a serverless compute service (FaaS) executes the code, what are lifecycle of function and different invocation types. Model description is based on Serverless White Paper published by CNCF Serverless Working Group [2]. Additionally, section 2.2.2 provides a description of FaaS implementation - a high-level description of how AWS Lambda operates.

2.2.1 Serverless Processing Model

The key elements of the serverless processing model are FaaS Controller, Event sources, Function instances, and Platform services. Figure 3 shows interaction between them.

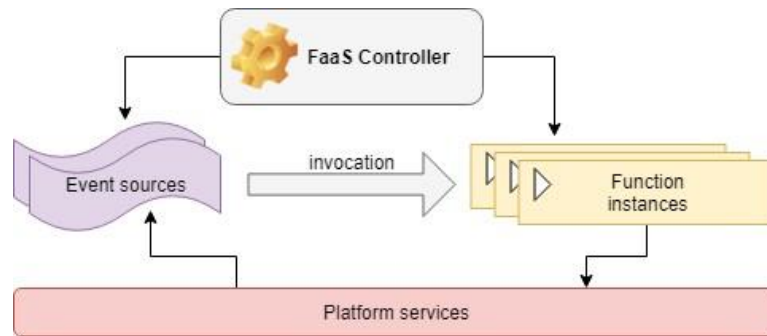


Figure 3 Key elements of FaaS solution

FaaS Controller is responsible for managing and monitoring function instances. Managing includes executing code, scaling instances depending on demand and terminating them when idle. Event sources emit, or stream events and event can trigger one or more function instances. Also, functions can be executed synchronously or asynchronously calling Invoke operation via AWS Lambda SDK. Such method when Lambda is invoked right away is called Push model. Another Pull model is used in streams, which batches new records together before invoking Lambda.

The executed function receives event data and context information as inputs. Event data depends on event source, for example, it can be an S3 event that an object has been stored and includes object Id, path and time. Context provides information about resources (e.g., memory/time limits) and execution environment (e.g., global and environment variables).

The function can communicate with platform services (BaaS) and they can emit new events.

2.2.2 Container Lifetime

Each function instance is executed in a container (sandboxed environment) to isolate them from one another and provide resources such as memory or disk. The first execution of function always takes longer time due to need of spinning up the container and starting a function instance. This process is called "cold start." According to AWS Lambda Overview and Best Practices document [3] initiation speed also depends on language, code package size, and network configuration. Reducing the size of code package helps to speed up initial execution because the package is downloaded from S3 each time before creating function instance. Best practices [3] suggests using interpreted languages instead of compiled ones to reduce the time of cold start.

When the function finishes execution, a container is suspended, and it is reused if the same function is triggered again. Background threads or other processes, spawned before the suspension, are resumed [4] too.

A container is terminated when it stays idle for a longer period of time. AWS does not provide official information about how long it keeps inactive containers. However Yan Cui carried experiments and concluded [5] that most of the time container is terminated after 45–60 minutes of inactivity, although sometimes it is terminated earlier. Frederik Willaert analyzed lifecycle of AWS Lambda containers and wrote an article [6] which includes intriguing observation, that

requests are always assigned to the oldest container. This implies that when containers are idle newly created containers are terminated sooner. Tim Wagner, general manager of AWS Lambda, revealed more interesting details in [6] article comment:

- a container can be reused up to 4-6 hours
- a container is considered as a candidate for termination if it has not been used in the last five minutes
- quickly abandoned containers are primary candidates for termination
- high levels of recent concurrency are scaled-down incrementally, in conjunction with policies that pack work into fewer containers at low rates of use.

The serverless function is stateless in theory, however, in practice it has state and that state can be used to increase performance, for example, keeping the database connection open between function invocations. Additionally, serverless providers are working to reduce cold-start frequencies and time.

3. AMAZON WEB SERVICES

AWS offers a wide range of managed services for building applications or processing data without owning servers. Using them instead of building a custom solution might greatly reduce development costs and time. Therefore, it is necessary to get familiar with provided services before developing something new. AWS serverless services are shown in Figure 4.

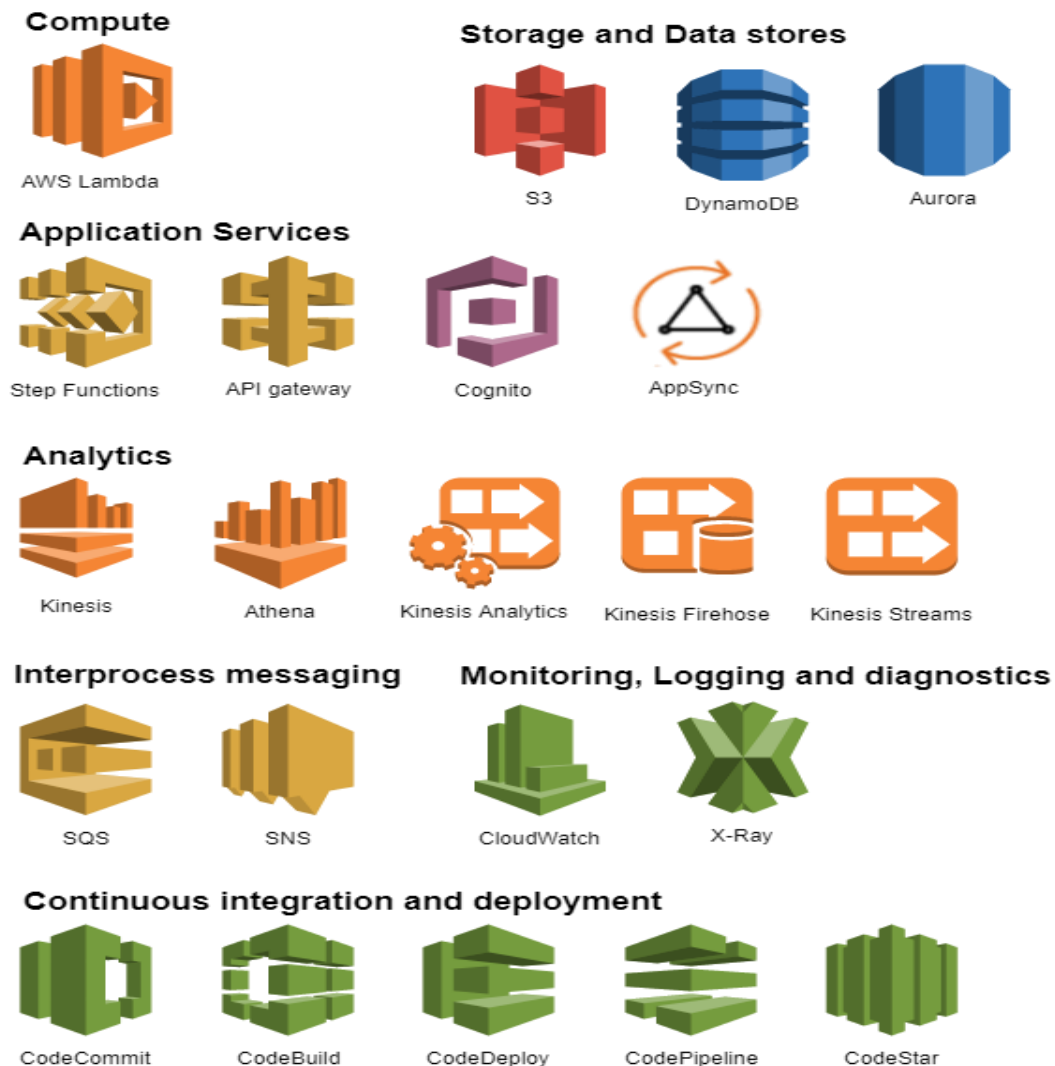


Figure 4 Amazon Serverless platform

The core of AWS serverless computing is *AWS Lambda*. This service lets execute code without provisioning servers. Lambda can be invoked by an event from other services and manually using AWS SDK or AWS console. The price depends on execution time and available RAM.

AWS Step Functions is a service to orchestrate the components of distributed applications using visual workflows. This service can be used to implement distributed transactions, coordinate

complex business logic, manage long running business processes and handle errors with built-in retry and fallback.

The price depends on a number of state transitions.

Amazon API Gateway is a service to create, maintain, secure and monitor REST API. The backend can be either EC2 instance or AWS Lambda. API Gateway allows invoking AWS Lambda function through HTTP call. Additionally, client SDK can be generated based on API definition for JavaScript, iOS, and Android. The price depends on a number of requests to API plus the cost of data transfer in and out.

AWS AppSync – real-time data synchronization layer via GraphQL. It can expose data from multiple sources such as DynamoDB table, Elasticsearch, and AWS Lambda. AWS AppSync can be used to build social media, chat, and collaboration applications. It offers real-time synchronization with built-in offline support, conflict resolution in the cloud and flexible security model. The price depends on a number of operations, connected time (real-time updates) and the amount of transferred data.

Amazon Cognito provides identity and access control services. It allows users to sign-up, sign-in using identities in Cognito User pools and federated users via social and enterprise identity providers. Security features include role-based access control, multi-factor authentication. Moreover sign-up, sign-in flow can be augmented using AWS Lambda triggers. The price depends on monthly active users.

AWS offers multiple serverless services to store data persistently.

Amazon Simple Storage Service (Amazon S3) – exposes secure, durable, highly-scalable object storage via simple REST API. The object can be any type, and any amount of data and objects can be grouped in buckets. Access to data is controlled per object and bucket policies. By default, all data is private. The price depends on the amount of stored and transferred data plus a number of requests. Data in Amazon S3 can be analyzed running SQL queries using Amazon Athena. The pricing is based on the volume of scanned data.

Amazon DynamoDB – low-latency NoSQL database supporting document and key-value storing models. Access to data can be configured per field granularity. The price depends on the size of written, read, and indexed data per Capacity Unit (scalability measure).

Amazon Aurora Serverless - on-demand, auto-scaling configuration for Aurora. Amazon Aurora is a relational database which is compatible with MySQL and with PostgreSQL. Serverless configuration automatically start-up, scale up/down database based on load and shuts down when a database is not used.

All storage services except for Amazon Aurora have integration with AWS Lambda.

Streaming data can be handled using *Amazon Kinesis*. *Amazon Kinesis* is a group of services designed to collect, process, and analyze real-time, streaming data like application logs, IoT sensors data, audio, video, and other continuously generated data. *Kinesis Data Firehose* is for capturing, preparing and loading streaming data into data stores and analytics tools; the price depends on the volume of ingested data. *Kinesis Video Streams* is to ingest and process video streams for analytics and machine learning. The price is calculated based on the volume of ingested, stored and consumed. *Kinesis Data Streams* is service to process or analyze streaming data and price depends on Shard Hour (processing capacity) and PUT Payload Unit (25KB). *Kinesis Data Analytics* allows to process streaming data using SQL queries and pricing is calculated by

Kinesis Processing Unit (processing capacity).

AWS has two services for inter-process messaging.

Amazon Simple Queue Service (SQS) is a fully managed message queuing service. There are two types of message queues. Standard queues guarantee atleast-once delivery and give maximum throughput. SQS FIFO guarantees exactly once delivery in the exact order that they are sent, however, throughput is limited to 300 transactions per second. The price depends on a number of requests and type of queue, plus bandwidth.

Amazon Simple Notification Service (SNS) is a fully managed publish/subscribe messaging service. Published messages can be pushed to Amazon Simple Queue Service (SQS) queues, AWS Lambda functions, and HTTP endpoints. The price depends on the number of published and delivered notifications.

AWS allows having whole continuous integration pipeline without managing any servers using AWS Code services which are described below.

AWS CodeCommit – is a fully-managed source control service to host private Git repositories. Pricing is based on a number of active users who access the repository during the month. AWS services like AWS CodeBuild and AWS CodePipeline which access repositories are counted as an active user. *AWS CodeBuild* provides service to compile source code, run tests and produce artifacts to deploy. Builds are processed in parallel, therefore, no waiting queue. The price is calculated per build duration and capacity of build instance. *AWS CodeDeploy* provides automatic deployment service to EC2 instances and on-premises servers. Services are updated gradually, via configurable rules deployment is monitored and in case of error deployment can be stopped, and changes rolled back. This service costs only deploying to on-premises and price is calculated per number of updated instances.

AWS CodePipeline is a continuous integration and continuous delivery service. The pipeline is highly customizable and can be integrated with various thirdparty services not only AWS. For example, source code can be stored in GitHub, Jenkins can be used to build, Apica to test and XebiaLabs to deploy. There is a fixed monthly \$1 charge per active pipeline for using this service.

AWS CodeStar is a service to provide an entire continuous delivery platform for simple projects. Code changes are automatically built, tested, and deployed using AWS CodeCommit, AWS CodeBuild, AWS CodePipeline, and AWS CodeDeploy. Project templates help to setup deployment infrastructure for AWS Lambda, Amazon EC2, or AWS Elastic Beanstalk. This service is very easy to use, however, it not so advance and flexible as AWS CodePipeline. AWS CodeStar is free, only services used by its costs.

AWS has two services to monitor serverless applications - AWS X-Ray and Amazon CloudWatch.

Amazon CloudWatch – is a general monitoring service for AWS services. The service allows collect and track metrics and logs, set up alarms and react to any change in AWS resources. For example, CloudWatch Events can invoke AWS Lambda function in the response of the change in Amazon EC2 instance, like connect to the instance and install necessary software when a new instance with a specific tag is started. CloudWatch provides configurable dashboards for metrics visualization and logs explorer with filtering functionality. The price depends on the number of dashboards, number of custom metrics, number of alarms, the volume of logs and number of custom events.

AWS is a cloud service platform, providing a plethora of services for the development software packages and varied applications. The developer is not responsible and receives hassle free solutions for accessibility, availability, scalability, security, redundancy, virtualization, networking or computing as Amazon handles them automatically looking into the requirements.

3 3.1 Simple Storage Service (S3)

Amazon S3 in simple terms is built as a storage for the Web. It is tailored to enable easier web-scaled computing for the developing community. It has a simple interface that is used for storage and retrieval any amount of data, at any time, universally on the web. Fundamentally, it is a key blog store. Each created S3 object has data, a key, and metadata. The key name uniquely identifies the object in a bucket. Object metadata is a set of name-value pairs and is set at the time it is uploaded. Blogs are associated with unique key names which makes them easier to sort and access. They may contain information such as Meta data (e-tag), creation time information etc. Amazon S3 is extremely durable.

3.2 Lambda (λ)

AWS Lambda is a trigger-driven computing cloud platform that allows programmers to code functions on a pay-as-you-go basis without having to define storage or compute resources. One of the most prominent advantages of AWS Lambda is that it uses abstraction and segregates server management from end user. With its use, Amazon itself handles the servers, which enables a programmer to focus more on writing code. Lambda follows the Function-as-a-Service (FaaS) model and allows any arbitrary function to run on AWS in a wide range of programming languages including Node.js, Python, Java and C#. Users can also utilize code compiler tools, such as Gradle or Maven, and packages to build functions on a serverless platform.

Why Lambda over other Services?

Lambda enables the creation of a new execution of functions in milliseconds. The time taken to execute a function, factors in Lambda. Comparing with other services for instance EC2 (Elastic Compute Cloud) wherein a server boots up the OS before doing any work, Lambda bills a user for just the time you took rounded up-to to the next 100ms and bill is generated for just the power computed. On the other hand, EC2 bills by the second. Servers are billed till the time they are shut down by the user. The user also might be forced to pay for the computing power he/she might not be using.

For Lambda, all you need to do is code, build and upload your function. And from there, triggers will execute it however it is required. In EC2, you have to pay for the servers OS boot time also. For example, if a user has a fast Lambda function, the bill is generated for 100ms depending on how fast the Lambda function is. But for the same function, EC2 can charge for the minute. For scaling, Lambda can scale up to a thousand parallel executions. In EC2, a user can scale only by adding more servers. A user might wind up paying for the CPU time they might not use. It is just not as granular at how well you can scale the servers which means you end up wasting money. In lambda, AWS automatically keeps the server up to date. In EC2 you can restart the system 4 A. Dani, C. Pophale, A. Gutte, B. Choudhary, and S.S. Sonawani with the updated image but that still takes time and has to be done explicitly. For all above discussed reasons, Lambda is chosen as an integral framework for the application.

3.3 Identity Access Management (IAM)

IAM manages users and groups for the user. It defines "roles" which allows a user to define pre-packaged access sets. There is a functionality to assign roles to users and we also needed to assign roles to our Lambda function and your API gateway as we built our serverless application [13]. Policies in IAM are the actual access control descriptions. A developer adds a policy to the S3 bucket to actually provide public access to that bucket. This is where the details actually live. Policies can't do anything on their own. They need to be assigned to a role or a user or a group in order to do something. We need to attach the policy to that entity in order to underline how we interact with that thing. It also manages identity providers and user account settings. Encryption

keys are something that AWS can manage through IAM and it will generate and provide access to encryption keys.

3.4 DynamoDB

DynamoDB is a NoSQL database provided by AWS. It is basically a key-value storage system. It does not typically provide ACID (atomicity, consistency, isolation, durability). There is a really good chance that data might be written but may be not immediately. For retrieving items from DynamoDB:

- Get Item: It is the easiest way to retrieve an item. User just has to provide a key to use it. The result will be a single item or if the item does not exist, it will through an error.
- Query: Queries require a hash key. If the user does not have a sort key, there is no need to do a query. User can do a "get item" with a hash key. The client can also put a constraint on sort key for retrieving a particular set of data.
- Scan: If you don't know a key, you can use a scan. It is a brute force approach in which the entire dataset is scanned. A filter can be applied to retrieve particular information from the dataset.

In DynamoDB, a user needs to set different capacities for read and write functionalities. DynamoDB gives an option of how much consistency is needed but the prices get increased gradually.

- 1 read unit = 1 consistent read up to 4 kb per second.
- 1 write unit = 1 write up to 1 kb per second.

3.5 API Gateway

Amazon API Gateway is a service offering that allows a developer to deploy non-AWS applications to AWS backend resources, such as servers. This allows two or more programs to communicate with each other to achieve greater efficiency. A user creates and manages APIs within API Gateway, which receives and processes concurrent calls. A developer can connect to other services such as EC2 instances, AWS Elastic Beanstalk and code from AWS Lambda. In order to create an API, a developer defines its name, an HTTP function, how the API integrates with services and how requests and transfers are handled. It accepts all payloads including JSON (JavaScript Object Notation) and XML (Extensible Markup Language). An AWS user can monitor API calls on a metrics dashboard in Amazon API Gateway and can retrieve error, access and debug logs from Amazon's CloudWatch. The service also allows an AWS user to maintain different versions of an API concurrently.

3.6 Cognito

Amazon Cognito is a User Management System used to manage user pools that controls user authentication and access for mobile applications. It saves and synchronizes client data, which enables an application programmer to focus on writing code instead of building and managing the back-end infrastructure. Amazon Cognito collects a user's profile attributes into directories referred as user pools and associate's information sets with identities and saves encrypted data as key-value pairs within Amazon Cognito sync storage.

3.7 CloudWatch and CloudFront

CloudWatch is used to provide data insights, graphical representation of resources used by the application. The dash boards provided by AWS for cloud watch has an umpteen option for data crunching. Cloud Watch is essential for resource monitoring and management for developers, system operators, site reliability engineers, and IT managers.

Amazon CloudFront is a middle-ware which resides in between a user request and the S3 data center in a specific region. CloudFront is used for low latency 6 A. Dani, C. Pophale, A. Gutte, B. Choudhary, and S.S. Sonawani distribution of static and dynamic web content from S3 to the user. Amazon Cloud Front is a Content Delivery Network (CDN) proxies and caches web data at edge locations as close to users as possible.

4. LITERATURE SURVEY

4.1 Improving Web Application Deployment in the Cloud

Roberts and Chapin's work is a decent beginning stage to get an inside and out perspective on the Serverless space, representing territories where the Serverless area needs improvement, for instance seller lock-in, state the board, the absence of any Serverless engineering examples, and that's just the beginning Lynn et al. look at FaaS contributions utilizing different models other than execution and cost. They contend that the promoted benefits of Serverless depend on scarcely any utilization cases and examination papers. In any case, Adzic and Chatley contemplated two creation applications that have effectively changed to Serverless and tracked down that the most convincing explanations behind progressing are facilitating costs. They additionally inferred that high-throughput applications are a preferred decision over high accessibility ones with regards to costs. Eivy thought about costs for running an application on a virtual machine as opposed to running it on Serverless and found that it is less expensive to send it on virtual machines if there is a steady and unsurprising burden. He prompts that thorough testing and recreations ought to be performed in any case, prior to choosing to move to Serverless. Trihinas et al. put forth the defense for microservices reception, representing their disadvantages and introducing an answer that vows to tackle the current difficulties. Georgiou et al. explored a particular usecase, Internet-of-Things applications and edge preparing, while displaying their system for streaming sensor examination with the assistance of questions. Difficulties of current applications and effectively address these in one manner or another, with the utilization of microservices and holders. This demonstrates that albeit serverless may address similar issues, there are strong other options, like microservices and holders, that ought to be thought of or stunningly better, be utilized along with serverless.

4.2 Serverless Computing Performance

Back and Andrikopoulos as well as Lee et al. did performance testing to compare FaaS offerings of different providers. They did this by gradually incrementing load on a single function while observing resource usage and comparing costs. The benchmark they defined can be used when individual raw power of a function is concerned. Lee et al. on the other hand, tracked more than just resource consumption and concluded that distributed data applications are a good candidate for FaaS, whereas applications that require high-end computing power are not a good fit. They found that the main reasons for this are the known execution time limit and fixed hardware resources. Lloyd et al. deployed microservices as FaaS and looked at a variety of factors that affect their performance. They did this by changing the load in order to observe how the underlying infrastructure performs. Some papers look at possible use-case for Serverless by considering their advertised benefits and costs. Others investigated FaaS performance in-depth and across multiple cloud providers. We see an opportunity here to investigate a specific and common use-case for FaaS, namely Web APIs. We plan to focus on the end-user's perspective by measuring and comparing a single metric, i.e., response times.

5.METHODOLOGY OF THE PROJECT

The showcasing term 'serverless' alludes to another age of stage as-a-administration contributions by significant cloud suppliers. These new administrations were led by Amazon Web Services (AWS) Lambda, which was first reported toward the finish of 2014 [7], and which saw critical reception in mid to late 2016. All the significant cloud specialist co-ops currently offer comparative administrations, like Google Cloud Functions Azure Functions³ and IBM OpenWhisk⁴. This paper essentially talks about AWS Lambda, as this was the primary stage to dispatch and is the most completely included Beyond the specialized comfort of lessening standard code, the financial aspects of AWS Lambda charging fundamentally affect the engineering and plan of frameworks. Past investigations have shown decreases of expenses in research center trials – here we analyze the impacts during modern application. We talk about three fundamental factors that we have noticed influencing structural choices when serverless figuring is free. In view of this parcel key, DynamoDB store information in various drives.

A proficient dispersion will make getting to the information as quick as could be expected, so it's imperative to pick a decent segment key. Along these lines, the parcel key can turn into the essential key, yet you can likewise utilize a mix of a segment key and a sort key as an essential key. Serverless is a blend of 'Capacity as a Service' and 'Backend as a Service. At undeniable level 'Stage as a Service' seems to be like serverless methodology, nonetheless, it isn't. According to Adrian Cockcroft, "If your PaaS can productively begin occasions in 20ms that run for a large portion of a second, at that point call it serverless." PaaS stage isn't adaptable as FaaS does. Serverless resembles a public vehicle. Use it and pay only for the utilization. It can scale also.

6. IMPLEMENTATION OF THE PROJECT

The diagram below provides a visual representation of the services used and how they are connected. This application uses AWS S3, Amazon API Gateway, AWS Lambda, Amazon DynamoDB, and AWS Identity and Access Management (IAM).

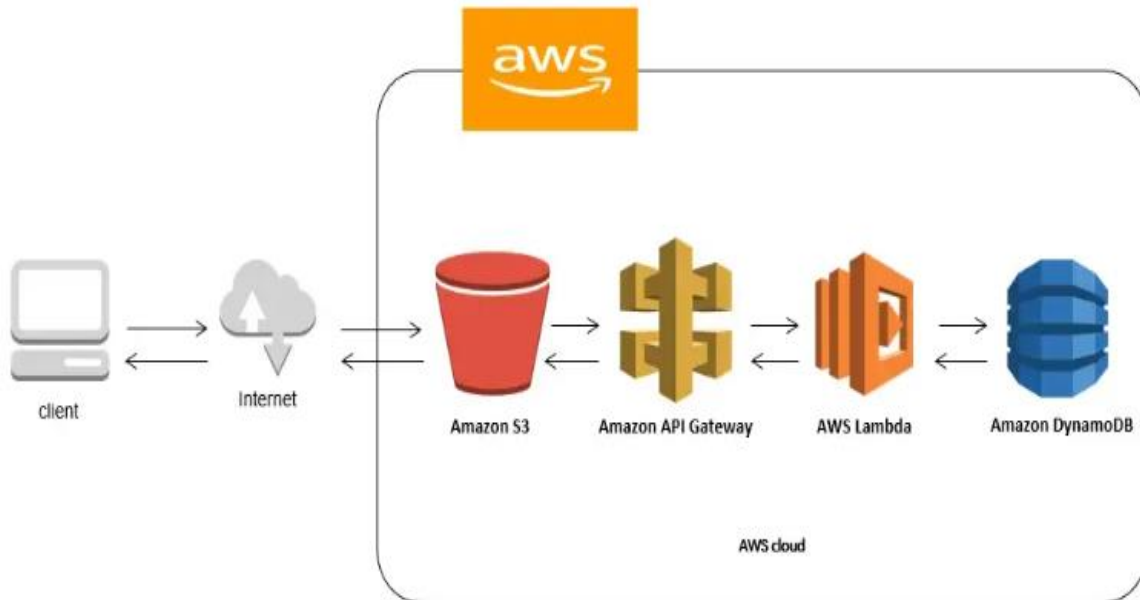


Figure 6.1 Architecture of the project

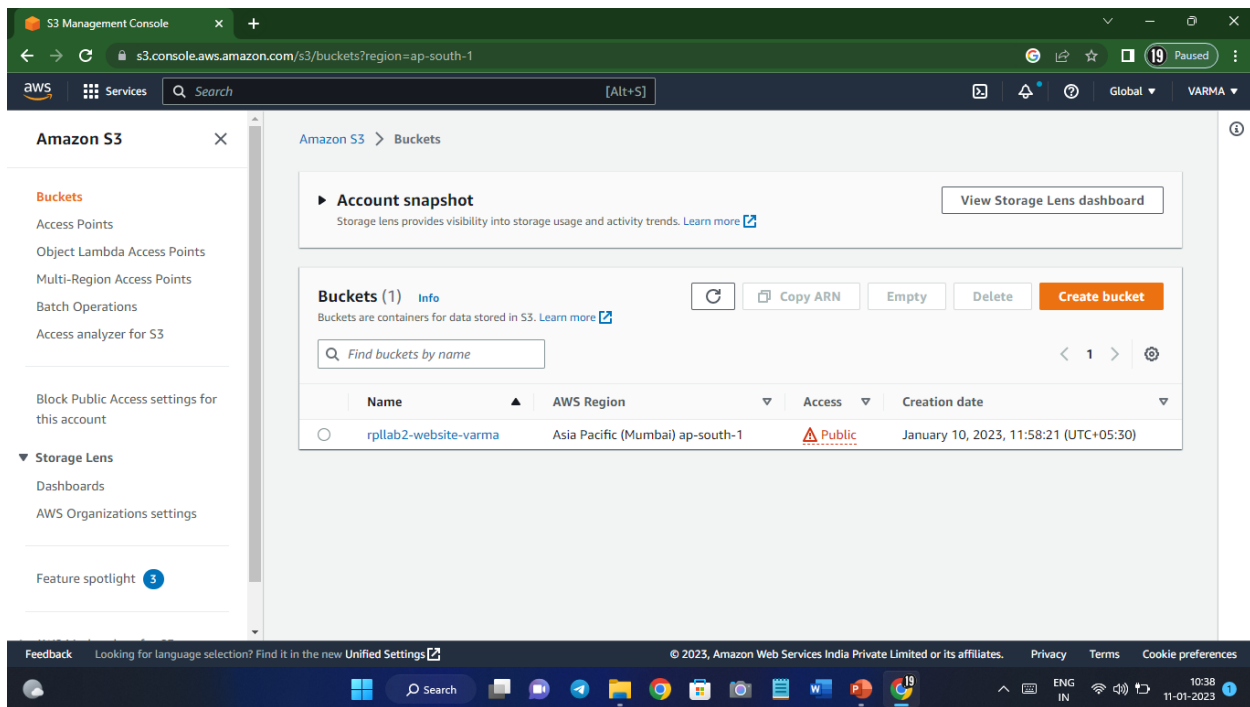
The implementation of the website involves these steps:

1. Creation of web Application
2. Build a serverless function
3. Linking of serverless function to web app
4. Create a table in database
5. Adding interactivity to website

6.1 Creation of Web Application

i. Create a web app with S3 console

- a. Create a s3 bucket named 'website-<yourname>'



- b. Upload the index.html & error.html into the bucket

Index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Hello World</title>
</head>

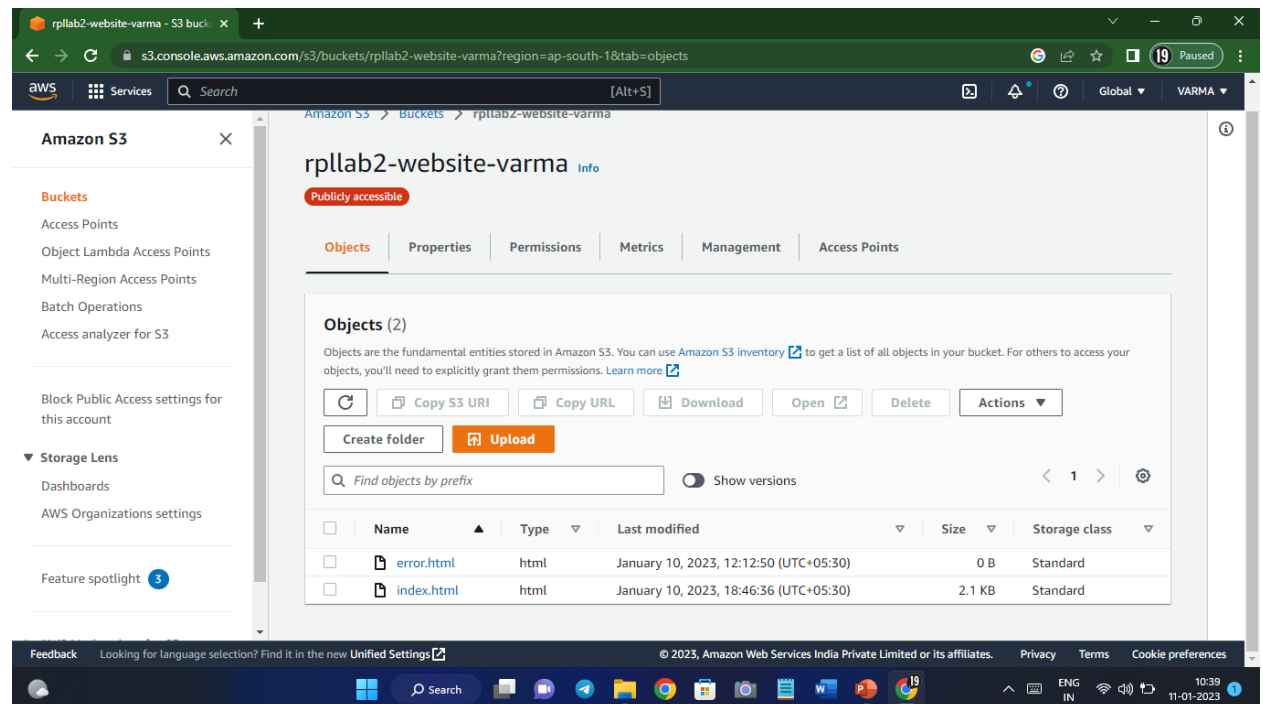
<body>
  Hello World
</body>
</html>
```

Error.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
```

```
<title>Error</title>
</head>

<body>
  Oops!! The page cannot be loaded
</body>
</html>
```



- c. Navigate to the s3 bucket properties and enable static website hosting
- d. Go back to Objects, select index.html & error.html and make it public

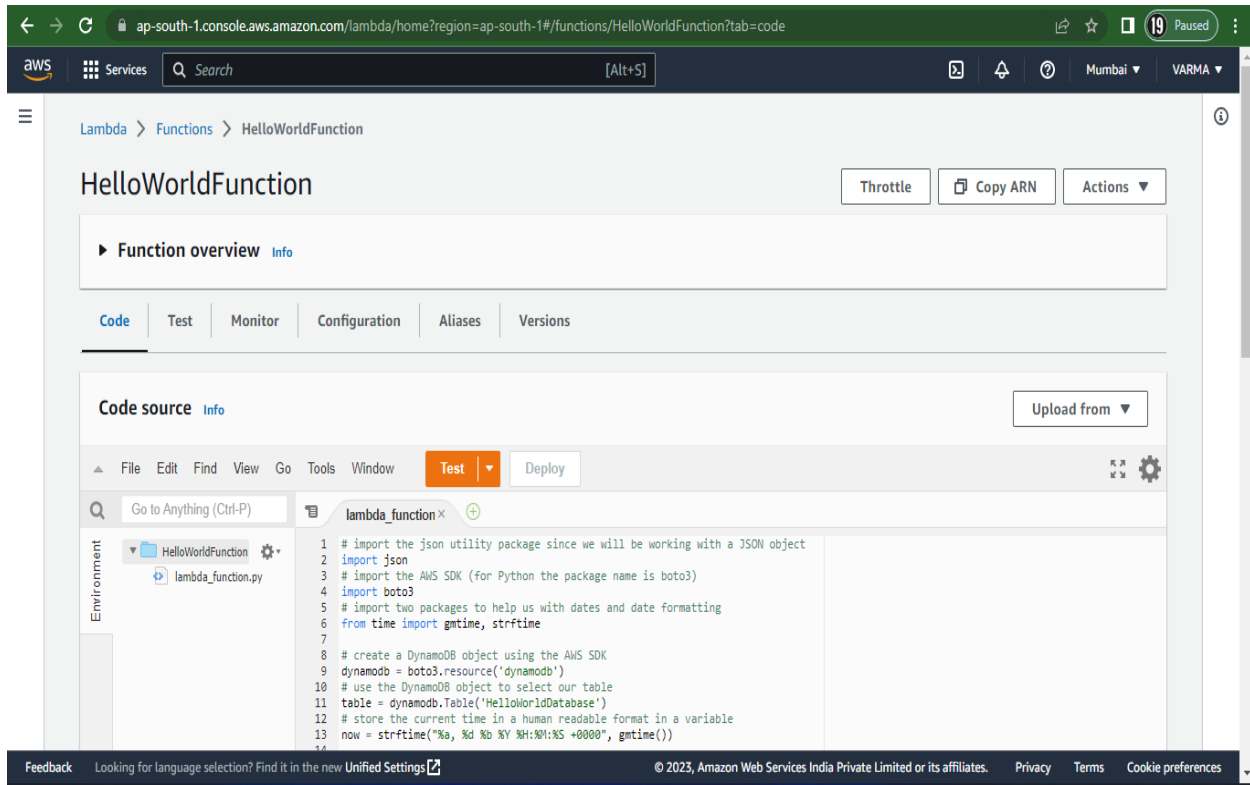
ii. Test your web app

Go to the properties and to Static website hosting and click on the link, it will open an interface then we can confirm that it is working.

6.2 Build a Serverless Function

i. Create and configure Lambda Function

- a. In a new browser tab, log in to the [AWS Lambda console](#).
- b. Make sure you create your function in the same Region in which you created the web app in the previous module. You can see this at the very top of the page, next to your account name.
- c. Choose the orange Create function button.
- d. Under Function name, enter *HelloWorldFunction*.



- e. Select Python 3.8 from the runtime dropdown and leave the rest of the defaults unchanged.
- f. Choose the orange Create function button.
- g. You should see a green message box at the top of your screen with the following message "Successfully created the function HelloWorldFunction."
- h. Under Code source, replace the code in lambda_function.py with the following:

```
import json

def lambda_handler(event, context):

    name = event['firstName'] + ' ' + event['lastName']

    return {

        'statusCode': 200,

        'body': json.dumps('Hello from Lambda, ' + name)

    }
```

- i. Save by going to the file menu and selecting Save to save the changes.
- j. Choose Deploy to deploy the changes.
- k. Let's test our new function. Choose the orange Test button to create a test event by selecting Configure test event.

l. Under Event name, enter *HelloWorldTestEvent*.

m. Copy and paste the following JSON object to replace the default one:

```
{
  "firstName": "Ada",
  "lastName": "Lovelace"
}
```

n. Choose the orange Create button at the bottom of the page.

ii. Test the Lambda Function

- a. Under the HelloWorldFunction section at the top of the page, select Test tab.
- b. You should see a light green box at the top of the page with the following text: *Execution result: succeeded*. You can choose Details to see the event the function returned.

The screenshot shows the AWS Lambda console interface for the 'HelloWorldFunction'. The 'Test' tab is selected, and the 'Execution results' section is expanded, showing a successful test event. The console includes a top navigation bar with the AWS logo, a search bar, and a 'Services' menu. The main content area has tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Test' tab displays a 'Code source' section with an 'Upload from' button and a 'Test' button. Below this, the 'Execution results' section shows the 'Test Event Name' as 'HelloWorldTestEvent' and the 'Response' as a JSON object: `{ "statusCode": 200, "body": "\"Hello from Lambda, Varma Paloji\"" }`. The 'Function Logs' section shows the execution details, including the 'Request ID', 'Duration', 'Billed Duration', 'Memory Size', and 'Max Memory Used'.

6.3 LINKING SERVERLESS FUNCTION TO WEB APP

i) Create new REST API

- Log in to the [API Gateway console](#).
- In the Choose an API type section, find the REST API card and choose the Build button on the card.
- Under Choose the protocol, select REST.
- Under Create new API, select New API.
- In the API name field, enter *HelloWorldAPI*.
- Select Edge optimized from the Endpoint Type dropdown. (Note: Edge-optimized endpoints are best for geographically distributed clients. This makes them a good choice for public services being accessed from the internet. Regional endpoints are typically used for APIs that are accessed primarily from within the same AWS Region.)
- Choose the blue Create API button. Your settings should look like the accompanying screenshot.

The screenshot shows the 'Create new API' form in the Amazon API Gateway console. It is divided into three sections: 'Choose the protocol', 'Create new API', and 'Settings'.

- Choose the protocol:** A heading followed by the instruction 'Select whether you would like to create a REST API or a WebSocket API.' Below this are two radio buttons: 'REST' (selected) and 'WebSocket'.
- Create new API:** A heading followed by the instruction 'In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.' Below this are four radio buttons: 'New API' (selected), 'Clone from existing API', 'Import from Swagger or Open API 3', and 'Example API'.
- Settings:** A heading followed by the instruction 'Choose a friendly name and description for your API.' Below this are three fields:
 - 'API name*': A text input field containing 'HelloWorldAPI'.
 - 'Description': An empty text input field.
 - 'Endpoint Type': A dropdown menu with 'Edge optimized' selected. An information icon (i) is to the right of the dropdown.

At the bottom left, there is a note '* Required'. At the bottom right, there is a blue button labeled 'Create API'.

ii) Create a new resource and method

- In the left navigation pane, select Resources under API: HelloWorldAPI.
- Ensure the "/" resource is selected.
- From the Actions dropdown menu, select Create Method.
- Select POST from the new dropdown that appears, then select the checkmark.
- Select Lambda Function for the Integration type.
- Select the Lambda Region you used when making the function (or else you will see a warning box reading "You do not have any Lambda Functions in...").
- Enter *HelloWorldFunction* in the Lambda Function field.
- Choose the blue Save button.
- You should see a message letting you know you are giving the API you are creating permission to call your Lambda function. Choose the OK button.
- With the newly created POST method selected, select Enable CORS from the Action dropdown menu.
- Leave the POST checkbox selected and choose the blue Enable CORS and replace existing CORS headers button.

Enable CORS

Gateway Responses for
HelloWorldAPI API

☐ DEFAULT 4XX ☐ DEFAULT 5XX ⓘ

Methods

☒ POST ☒ OPTIONS ⓘ

Access-Control-Allow-Methods

OPTIONS, POST ⓘ

Access-Control-Allow-Headers

'Content-Type,X-Amz-Date,Authorizatio ⓘ

Access-Control-Allow-Origin*

* ⓘ

▶ Advanced

Enable CORS and replace existing CORS headers

10. You should see a message asking you to confirm method changes. Choose the blue Yes, replace existing values button.

Confirm method changes

×

The following modifications will be made to this resource's methods and will replace any existing values. Are you sure you want to continue?

- Create **OPTIONS** method
- Add **200 Method Response** with **Empty Response Model** to **OPTIONS** method
- Add **Mock Integration** to **OPTIONS** method
- Add **200 Integration Response** to **OPTIONS** method
- Add **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Method Response Headers** to **OPTIONS** method
- Add **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Integration Response Header Mappings** to **OPTIONS** method
- Add **Access-Control-Allow-Origin Method Response Header** to **POST** method
- Add **Access-Control-Allow-Origin Integration Response Header Mapping** to **POST** method

Cancel

Yes, replace existing values

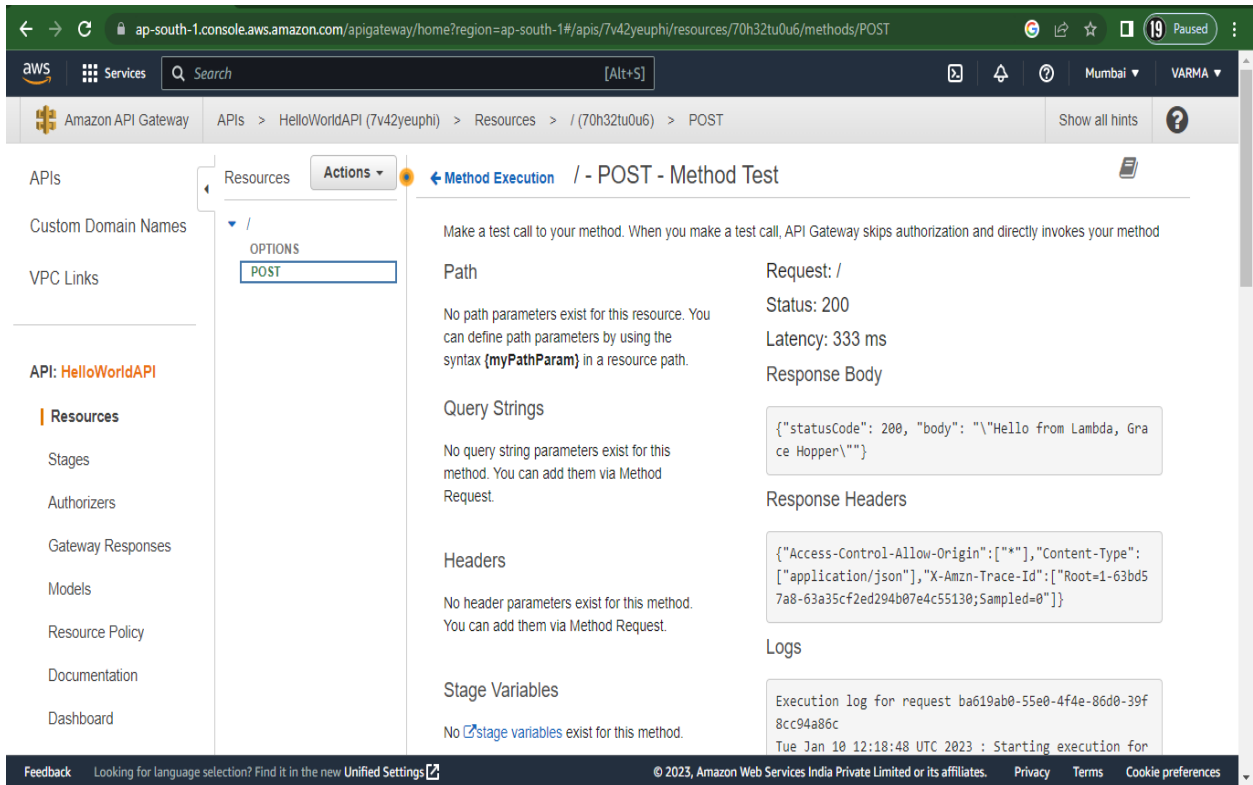
iii) Deploy API

- In the Actions dropdown list, select Deploy API.
- Select [New Stage] in the Deployment stage dropdown list.
- Enter *dev* for the Stage Name.
- Choose Deploy.
- Copy and save the URL next to Invoke URL (you will need it in module five).

iv) Validate API

- In the left navigation pane, select Resources.
- The methods for our API will now be listed on the right. Choose POST.
- Choose the small blue lightning bolt.
- Paste the following into the Request Body field:

```
{
  "firstName": "Grace",
  "lastName": "Hopper"
}
```
- Choose the blue Test button.
- On the right side, you should see a response with Code 200.
- Great! We have built and tested an API that calls our Lambda function.

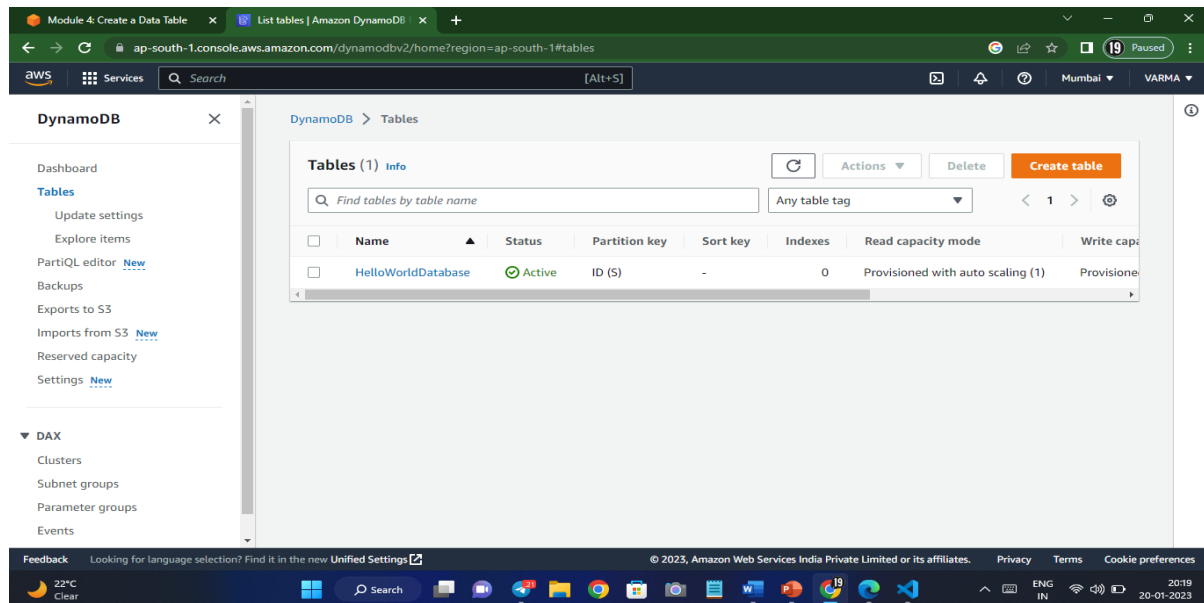


6.4 Create a DynamoDB table

i) Create a DynamoDB table

- Log in to the [Amazon DynamoDB console](#).
- Make sure you create your table in the same Region in which you created the web app in the previous module. You can see this at the very top of the page, next to your account name.
- Choose the orange Create table button.
- Under Table name, enter *HelloWorldDatabase*.
- In the Partition key field, enter ID. The partition key is part of the table's primary key.
- Leave the rest of the default values unchanged and choose the orange Create table button.

g. In the list of tables, select the table name, *HelloWorldDatabase*.



h. In the General information section, show Additional info by selecting the down arrow.

i. Copy the Amazon Resource Name (ARN). You will need it later in this module.

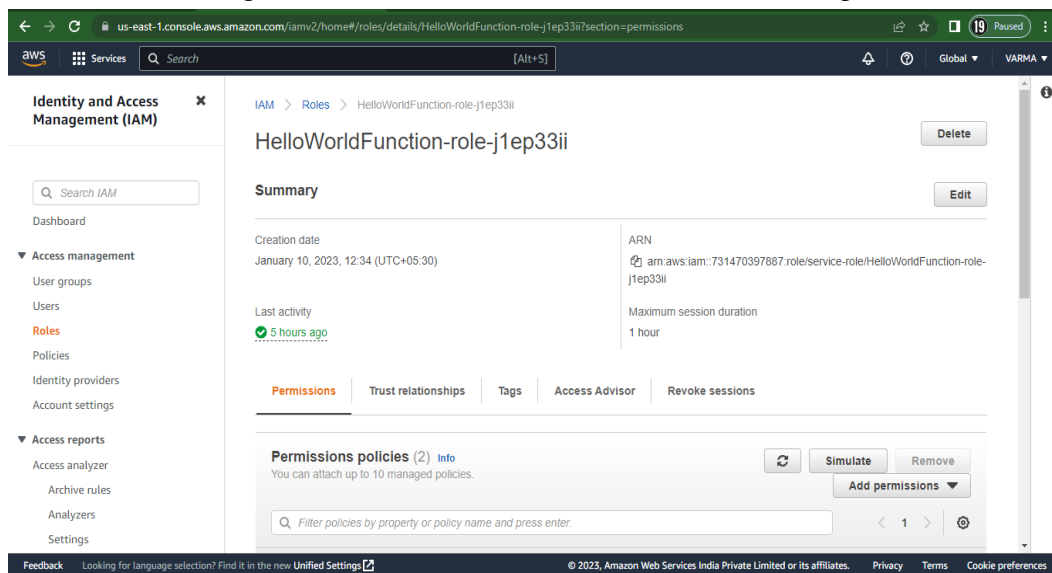
ii) Create and IAM policy to Lambda function

a. Now that we have a table, let's edit our Lambda function to be able to write data to it. In a *new* browser window, open the [AWS Lambda console](#).

b. Select the function we created in module two (if you have been using our examples, it will be called *HelloWorldFunction*). If you don't see it, check the Region dropdown in the upper right next to your name to ensure you're in the same Region you created the function in.

c. We'll be adding permissions to our function so it can use the DynamoDB service, and we'll be using AWS Identity and Access Management (IAM) to do so.

d. Select the Configuration tab and select Permissions from the right-side menu.



- e. In the Execution role box, under Role name, choose the link. A new browser tab will open.
- f. In the Permissions policies box, open the Add permissions dropdown and select Create inline policy.
- g. Select the JSON tab.
- h. Paste the following policy in the text area, taking care to replace your table's ARN in the Resource field in line 15:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb>DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:Scan",
        "dynamodb:Query",
        "dynamodb:UpdateItem"
      ],
      "Resource": "YOUR-TABLE-ARN"
    }
  ]
}
```

- a. This policy will allow our Lambda function to read, edit, or delete items, but restrict it to only be able to do so in the table we created.
- b. Choose the blue Review Policy button.
- c. Next to Name, enter *HelloWorldDynamoPolicy*.
- d. Choose the blue Create Policy button.
- e. You can now close this browser tab and go back to the tab for your Lambda function.

iii) Modify Lambda function to write to DynamoDB table

- a. Select the Code tab and select your function from the navigation pane on the left side of the code editor.
- b. Replace the code for your function with the following:

```
# import the json utility package since we will be working with a JSON object
import json
# import the AWS SDK (for Python the package name is boto3)
import boto3
# import two packages to help us with dates and date formatting
from time import gmtime, strftime

# create a DynamoDB object using the AWS SDK
dynamodb = boto3.resource('dynamodb')
# use the DynamoDB object to select our table
table = dynamodb.Table('HelloWorldDatabase')
# store the current time in a human readable format in a variable
```

```

now = strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())

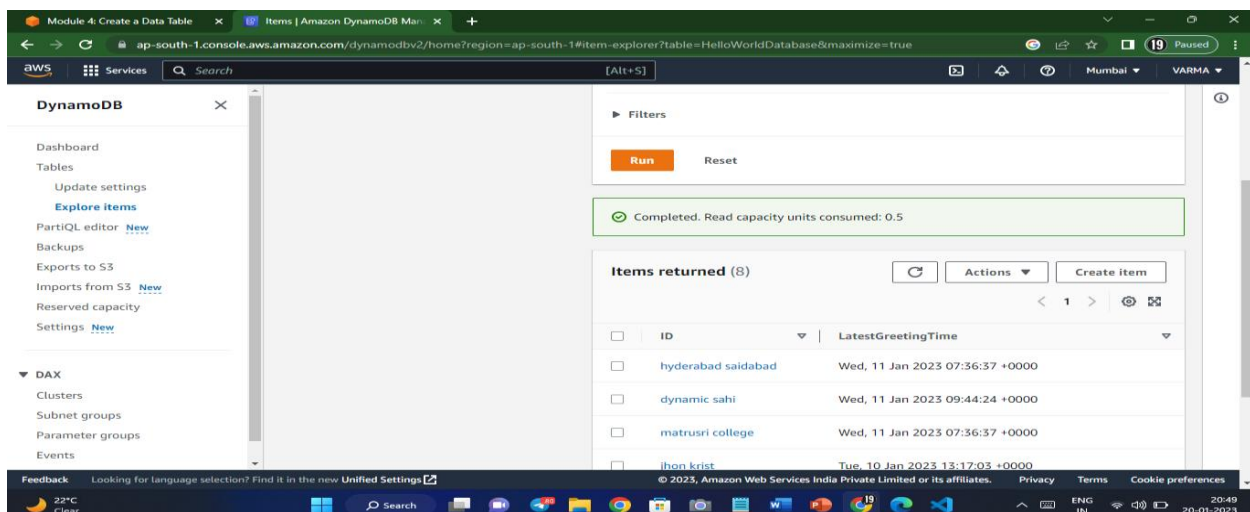
# define the handler function that the Lambda service will use as an entry
point
def lambda_handler(event, context):
    # extract values from the event object we got from the Lambda service and
    store in a variable
    name = event['firstName'] + ' ' + event['lastName']
    # write name and time to the DynamoDB table using the object we instantiated
    and save response in a variable
    response = table.put_item(
        Item={
            'ID': name,
            'LatestGreetingTime': now
        })
    # return a properly formatted JSON object
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda, ' + name)
    }

```

c. Choose the Deploy button at the top of the code editor

iii) Test the changes

- Choose the orange Test button.
- You should see an *Execution result: succeeded* message with a green background.
- In a new browser tab, open the [DynamoDB console](#).
- In the left-hand navigation pane, select Tables > Explore items.
- Select *HelloWorldDatabase*, which we created earlier in this module.
- Select the Items tab on the right.
- Items matching your test event appear under Items returned. If you have been using our examples, the item ID will be *Hello from Lambda, Ada Lovelace*.
- Every time your Lambda function executes, your DynamoDB table will be updated. If the same name is used, only the time stamp will change.



6.5 Add Interactivity to Web App

i) Update web app in s3 bucket

- Open the *index.html* file you created in module one.
- Replace the existing code with the following:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Hello World</title>
  <!-- Add some CSS to change client UI -->
  <style>
    body {
      background-color: #232F3E;
    }
    label, button {
      color: #FF9900;
      font-family: Arial, Helvetica, sans-serif;
      font-size: 20px;
      margin-left: 40px;
    }
    input {
      color: #232F3E;
      font-family: Arial, Helvetica, sans-serif;
      font-size: 20px;
      margin-left: 20px;
    }
  </style>
  <script>
    // define the callAPI function that takes a first name and last name as
parameters
    var callAPI = (firstName,lastName)=>{
      // instantiate a headers object
      var myHeaders = new Headers();
      // add content type header to object
      myHeaders.append("Content-Type", "application/json");
      // using built in JSON utility package turn object to string and
store in a variable
      var raw =
JSON.stringify({"firstName":firstName,"lastName":lastName});
      // create a JSON object with parameters for API call and store in a
variable
      var requestOptions = {
        method: 'POST',
        headers: myHeaders,
        body: raw,
        redirect: 'follow'
      };
      // make API call with parameters and use promises to get response
```

```

        fetch("YOUR-API-INVOKE-URL", requestOptions)
        .then(response => response.text())
        .then(result => alert(JSON.parse(result).body))
        .catch(error => console.log('error', error));
    }
</script>
</head>
<body>
    <form>
        <label>First Name :</label>
        <input type="text" id="fName">
        <label>Last Name :</label>
        <input type="text" id="lName">
        <!-- set button onClick method to call function we defined passing input
values as parameters -->
        <button type="button"
onclick="callAPI(document.getElementById('fName').value,document.getElementById('
lName').value)">Call API</button>
    </form>
</body>
</html>

```

- Make sure you add your API Invoke URL on Line 41 (from module three).
Note: If you do not have your API's URL, you can get it from the [API Gateway console](#) by selecting your API and choosing stages.
- Save the file.
- Upload the index.html file in s3 bucket
- After uploading the index.html file, make it public.

ii) Test the updated website

- Choose the URL under Domain.
- Your updated web app should load in your browser.
- Fill in your name (or whatever you prefer) and choose the Call API button.
- You should see a message that starts with *Hello from Lambda* followed by the text you filled in.

7. RESULTS

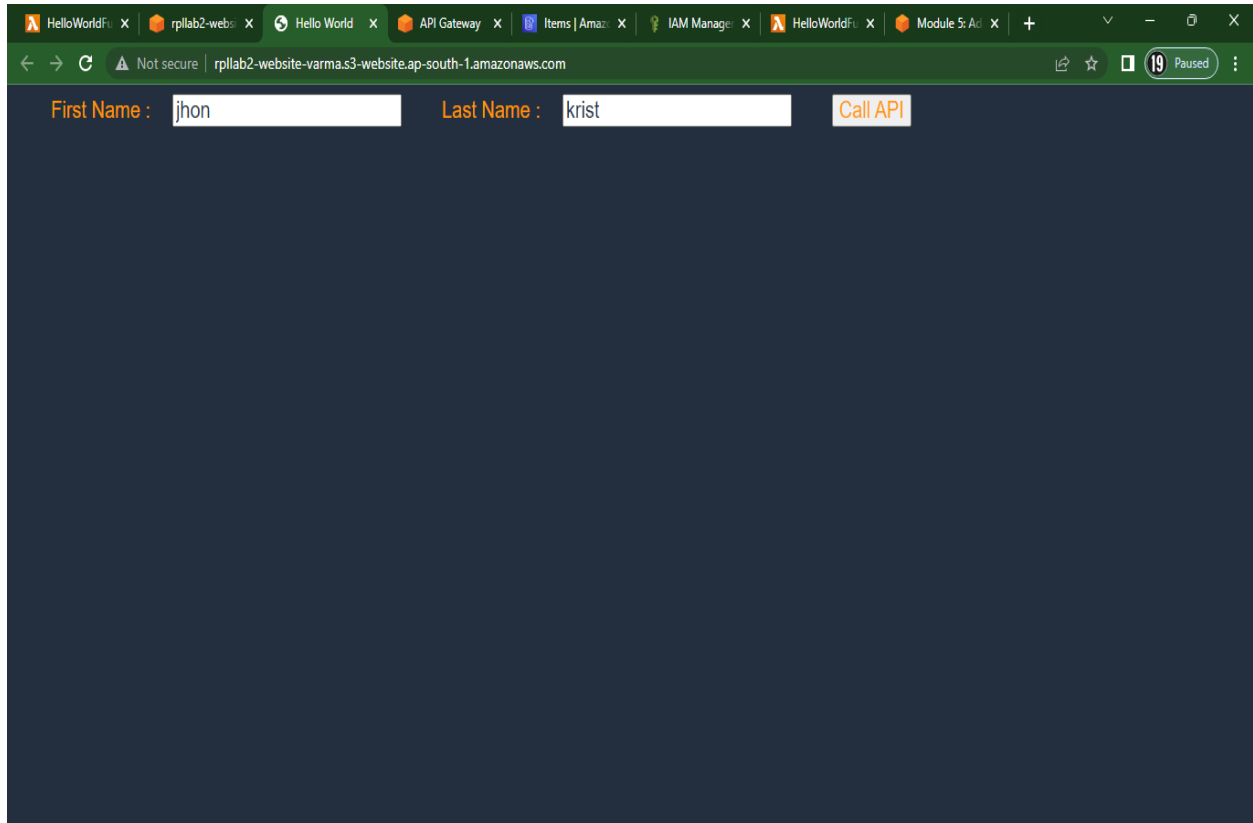


Figure 7.1: Website interface

After entering the details, you should see a message that starts with *Hello from Lambda* followed by the text you filled in.

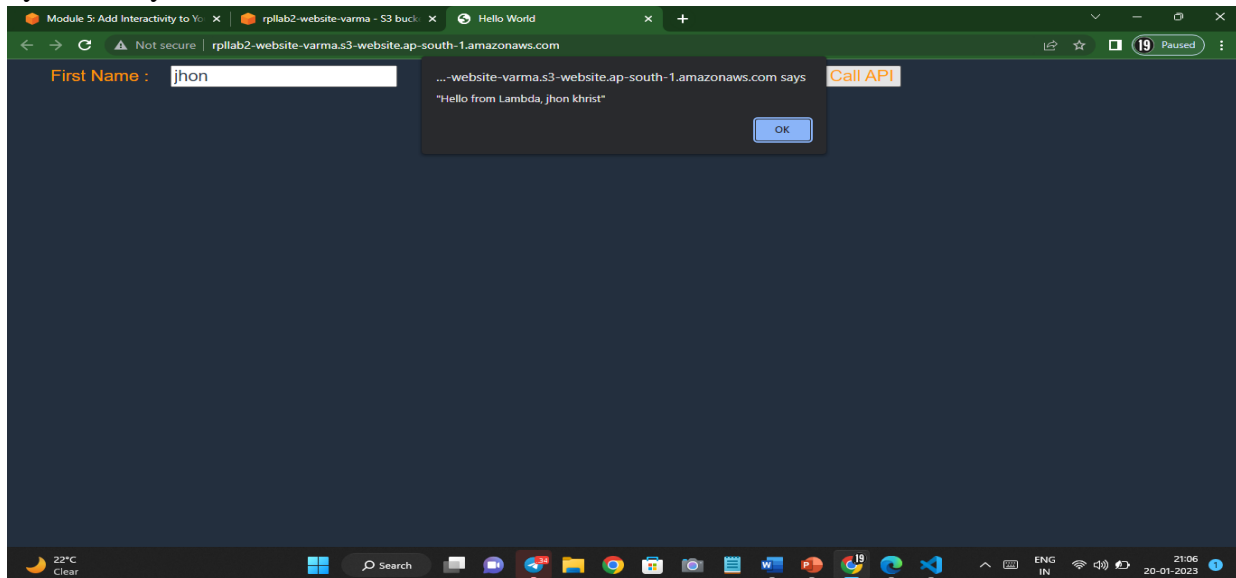
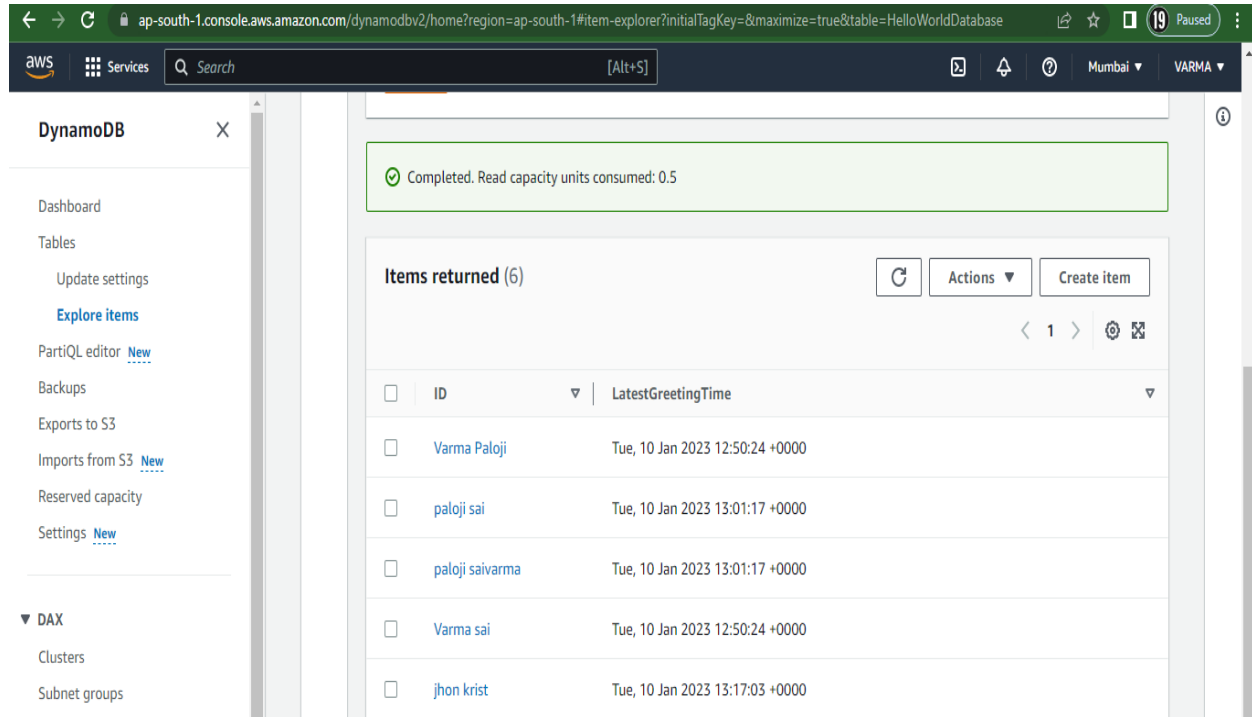


Figure 7.2: Message after entering the details

The details entered in the website are stored successfully in DynamoDB table.



The screenshot shows the AWS DynamoDB console interface. At the top, a green banner indicates 'Completed. Read capacity units consumed: 0.5'. Below this, a section titled 'Items returned (6)' displays a table with two columns: 'ID' and 'LatestGreetingTime'. The table contains six rows of data. The left sidebar shows the navigation menu with options like Dashboard, Tables, and Settings. The top navigation bar includes the AWS logo, a search bar, and the current region 'Mumbai' and account 'VARMA'.

ID	LatestGreetingTime
Varma Paloji	Tue, 10 Jan 2023 12:50:24 +0000
paloji sai	Tue, 10 Jan 2023 13:01:17 +0000
paloji saivarma	Tue, 10 Jan 2023 13:01:17 +0000
Varma sai	Tue, 10 Jan 2023 12:50:24 +0000
jhon krist	Tue, 10 Jan 2023 13:17:03 +0000

Figure 7.3: Details stored in DynamoDB

8. CONCLUSION

All in all, serverless stages today are valuable for significant (however not five-nines crucial) assignments, where high-throughput is critical, as opposed to low inertness, and where individual solicitations can be finished in a generally brief timeframe window. The financial aspects of facilitating such undertakings in a serverless climate make it a convincing method to lessen facilitating costs essentially, and to accelerate time to showcase for conveyance of new highlights. The possibility of a serverless application is fascinating: There's no compelling reason to deal with any workers. The application can be scaled consequently and exceptionally accessible. You pay just for the assets utilized and for the time the application is utilized. instructions to utilize AWS's API Gateway and Lambda capacities to assemble a REST API that performs CRUD procedure on an information base based on the AWS DynamoDB data set system. This provide additional information how to have this API on S3 in a solitary page application that can be circulated overall utilizing CloudFront. This was only a quick glance at every one of these advances; there is significantly more to find out about every innovation and about other AWS parts also.

WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES

1st w e e k	Date	Day	Name of the topic/Module Completed
	17/05/2022	Monday	Introduction, Cloud Architecture, Introduction to AWS, List of services
	18/05/2022	Tuesday	Management Service-IAM (Identity and Access Management), IAM Introduction-Users, Groups, Policies, IAM Users
	19/05/2022	Wednesday	Fundamentals of Networking, Networking in the Cloud, Bandwidth and Latency, IP Addressing Basics, Basics concepts of NAT
	20/05/2022	Thursday	Amazon Virtual Private Cloud (Amazon VPC), Amazon VPC Overview, Default and Non-Default VPC, Public and Private Subnets, Defining VPC CIDR Blocks
	21/05/2022	Friday	VPC – Continuation, Creating custom VPC with Subnets, Launch instances into subnets

2nd w e e k	Date	Day	Name of the topic/Module Completed
	24/05/2022	Monday	Amazon Elastic Cloud Compute (Amazon EC2) Basics: EC2 Pricing, How to SSH into EC2 using windows, Overview of EC2.
	25/05/2022	Tuesday	Basic concepts of Network and Security in EC2: 1) Elastic IPs, 2) Public & Private IP, 3) Placement Groups 4) Security Groups Overview
	26/05/2022	Wednesday	Overview AWS fundamentals RDS + Aurora + Elastic Cache Amazon RDS Overview, RDS Read Replicas vs Multi AZ
	27/05/2022	Thursday	Concepts of Aurora service, Overview of Elastic Cache, Assessment.
	28/05/2022	Friday	Sample Project Explanation.

9. REFERENCES

1. M. Sewak and S. Singh” Winning in the Era of Serverless Computing and Function as a Service,” 2018 3rd International Conference for Convergence in Technology (I2CT), Pune, 2018, pp. 1-5
2. T. Lynn, P. Rosati, A. Lejeune and V. Emeakaroha,” A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms,” 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, 2017, pp. 162-169.
3. M. Villamizar et al., ”Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures,” 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, 2016, pp. 179-182.
4. K. Swedha and T. Dubey, ”Analysis of Web Authentication Methods Using Amazon Web Services,” 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Bangalore, 2018, pp. 1-6.
5. W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly and S. Pallickara, ”Serverless Computing: An Investigation of Factors Influencing Microservice Performance,” 2018 IEEE International Conference on Cloud Engineering, Orlando, FL, 2018, pp. 159-169.
6. Z. Al-Ali et al., ”Making Serverless Computing More Serverless,” 2018 IEEE 11th International Conference on Cloud Computing, San Francisco, CA, 2018, pp. 456-459.
7. K. Kritikos and P. Skrzypek, ”A Review of Serverless Frameworks,” 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion, Zurich, 2018, pp. 161-168.
8. C. Kotas, T. Naughton and N. Imam, ”A comparison of Amazon Web Services and Microsoft Azure cloud platforms for high performance computing,” 2018 International Conference on Consumer Electronics, Las Vegas, NV, 2018, pp. 1-4.
9. H. Yoon, A. Gavrilovska, K. Schwan and J. Donahue, ”Interactive Use of Cloud Services: Amazon SQS and S3,” 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Ottawa, ON, 2012, pp. 523-530.
10. M. Villamizar et al., ”Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures,” 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, 2016, pp. 179-182.
11. P. Garca Lpez, M. Snchez-Artigas, G. Pars, D. Barcelona Pons, . Ruiz Ollobarren and D. Arroyo Pinto, ”Comparison of FaaS Orchestration Systems,” 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, 2018, pp. 148-153.
12. S. Narula, A. Jain and Prachi, ”Cloud Computing Security: Amazon Web Service,” 2015 Fifth International Conference on Advanced Computing & Communication Technologies, Haryana, 2015, pp. 501-505.

