Int.J.HIT.TRANSC:ECCN.Vol.10: Special Issue (2025) Page 91-95 ISSN: 0973-6875

Available Online at www.hithaldia.in/locate/ECCN

ORIGINAL CONTRIBUTION

Deployment of Serverless Web Applications Using Cloud Computing Technologies

[2]Souhardya De, [2]Subhajyoti Acharya, [2]Arijit Das Adhikary, [2]Aishwarya Maiti, [2]Ujjawal Singh, 2Sougata Ojha, [1]Jayanta Kumar Bag, [1]Ira Samanta, [1]Chanchal Kumar De

[1]Department of Electronics and Communication Engineering, Haldia Institute of Technology, Haldia, Purba Medinipur, West Bengal, Haldia Institute of Technology, Haldia, Purba Medinipur, West Bengal

[2,]UG student, Dept. of Electronics and Communication Engineering, Haldia Institute of Technology, Haldia, Purba Medinipur, West Bengal

ABSTRACT

The rapid evolution of web technologies has driven the adoption of serverless computing as a transformative approach to building and deploying modern web applications. Unlike traditional server-based architectures, serverless computing eliminates the need for developers to manage infrastructure, enabling them to focus solely on application logic. This paradigm significantly reduces operational complexity while optimizing costs through a pay-as-you-go model, where resources are utilized only when needed. Amazon Web Services (AWS), a leading cloud provider, offers a robust suite of services for efficient serverless application development. Key services such as AWS Lambda, Amazon API Gateway, Amazon S3, and Amazon DynamoDB empower developers to create scalable, resilient, and cost-effective applications without managing servers. These services simplify the deployment of dynamic web applications capable of handling unpredictable traffic patterns, ensuring high availability and reliability. By leveraging AWS serverless offerings, developers can streamline application development, minimize operational overhead, and adapt to fluctuating workloads with ease. This paper highlights the advantages of serverless computing, focusing on its ability to enhance scalability and reduce costs while maintaining optimal performance. Serverless architecture represents a significant shift in modern application development, enabling businesses to innovate and scale efficiently

KEYWORDS: Serverless computing, Amazon Web Services, Serverless Architecture, AWS Lambda, DynamoDB

1. INTRODUCTION

In the rapidly evolving digital landscape, businesses require applications that are scalable, cost-effective, and swiftly deployable. Traditional server-based architectures often struggle to meet these demands due to their complexity and the burden of infrastructure management. Serverless computing offers a solution by abstracting infrastructure management, allowing developers to concentrate solely on creating innovative and functional applications [1]. This work aims to explore serverless computing as a practical approach for modern application development and

*Corresponding Address: jkbag2008@gmail.com

DOI: 10.5281/zenodo.14592010

deployment, utilizing AWS services to fulfill these objectives. infrastructure management [2]. Small code snippets are provided to illustrate this automation. The primary objective of this thesis is to demonstrate and analyze the deployment of a serverless web application using Amazon Web Services (AWS) [3]. This involves leveraging serverless technologies to design, implement, and evaluate the performance, scalability, and cost-efficiency of a modern web application [4, 5].

HIGHLIGHTS OF THE WORK

i) Seamless Serverless Architecture Deployment:

By leveraging serverless technologies, this work demonstrates how to architect and deploy a fully functional web application without provisioning traditional servers. AWS Lambda is used to handle backend logic, API Gateway man- ages REST API requests, and DynamoDB provides a robust and scalable database solution. Automating this setup with AWS SDKs, as demonstrated below with a small function in boto3, allows seamless integration between components:
[language=Python] import boto3 client = boto3.client('apigateway') re- sponse = client.createrestapi(name =′ ServerlessAppAPI′, description='API for serverless web app')

ii) Infrastructure-as-Code with CloudFormation

Cloud Formation templates were created to define and manage AWS re- sources, allowing for consistent deployments and infrastructure version control. With a template, the infrastructure configuration is saved in code, which can be reused, modified, and tracked across various environments. Below is a basic ex- ample of Cloud Formation for deploying an S3 bucket:
[language=YAML] Resources: My Bucket: Type:'AWS::S3::Bucket' Prop- erties: Bucket Name:'my-serverless-app-bucket'

iii) Cost Efficiency and Scalability

Serverless architectures and on-demand instance provisioning allow for infrastructure that scales automatically based on load, ensuring cost-efficiency by charging only for the resources used. This approach optimizes resources for applications with variable or unpredictable workloads.

Rest of this paper is as follows: Section 2 briefly explains the roadmap method for deploying serverless application. Subsequently, section 3 describes the connection method between AWS lambda (serverless) and automate EC2 instances. In section 4, the method of installation of automating EC2 instances using AWS lambda is described precisely. The building of a serverless web application is illustrated in section 5. In section 6, the results and discussion are illustrated briefly. The conclusion of the paper is presented in Section 7.

2. ROADMAP FOR DEPLOYING SERVERLESS APPLICATION:

The roadmap for deploying serverless application is implemented in the following steps [7, 8]:

A. Configure IAM Role with Permissions for Lambda Functions and Dy- namoDB

- Create an IAM role with permissions to access Lambda functions and DynamoDB.

B. Create a DynamoDB Table

- Open AWS Management Console and navigate to **DynamoDB**.
- Click on **Create Table**.
- Specify the table name (e.g., *student_data*) and partition key (e.g.,*student_id*).
- Optionally, define a sort key (e.g.,M *student_name*).
- Click on **Create Table**.

C. Create Lambda Functions

• Open AWS Management Console and search for Lambda.

• Click on Create Function and select From Scratch.

- Provide a function name (e.g., get_student_data) and choose Python 3.9 as the runtime.
- Configure the execution role with permissions to access DynamoDB and click on Create Function.
- Repeat the steps to create an additional Lambda function (e.g., in- sert_student_data).

D. Create API Gateway

- Navigate to API Gateway in the AWS Management Console.

- Click on Create API and provide an API name (e.g., student_api).
- Choose Edge-optimized as the API endpoint type and click on Create API.

E. Create Methods

- In API Gateway, create a GET method for retrieving student data, configured to use the get_student_data Lambda function.
- Create a POST method for inserting student data, configured to use the insert_student_data Lambda function.

F. Deploy API

- In the API Gateway console, deploy the API to a stage (e.g., prod).

G. Host Web Application

- Create an S3 Bucket: Create a new S3 bucket to store your web application files.
- Upload Files: Upload your index.html and scripts.js files to the S3 bucket.
- Enable Static Web Hosting: Configure static web hosting for the S3 bucket.
- Update scripts.js: Replace the endpoint URL in scripts.js

with the API Gateway URL.

H. Update S3 Bucket Policy

- Configure the S3 bucket policy to grant CloudFront permission to access your objects.

I. Use CloudFront URL

- Replace the S3 bucket URL in your scripts.js file with the Cloud- Front URL for enhanced performance.

3. CONNECTING AWS LAMBDA (SERVERLESS) TO AUTOMATE EC2 INSTANCES

To launch an AWS Lambda function using the AWS CLI, follow these steps [9,10].

Step 1: Create a Deployment Package

• If the code is written locally, compress it into a zip file. For example, if

• There is a Python file named lambda_function.py, use: zip function.zip lambdaf unction.py.

Step 2: Create an IAM Role

• Create an IAM role for the Lambda function with the necessary permissions, either via the AWS Management Console or using AWS CLI.

Step 3: Write AWS CLI Command to Launch AWS Lambda

Use the following command to create the Lambda function:

[language=bash] aws lambda create-function–function-namedummy lambda1zipfilefileb:// lambdaunction.ziphandlerlambdahandler.lambdahandler–runtime python3.12–rolearn:aws: iam::248189908910:role/amitrole

4. AUTOMATING EC2 INSTANCES USING AWS LAMBDA

The installation of automating EC2 instances using AWS LAMBDA is executed in the following steps [11,12]:

Step 1: Access the AWS Lambda Function

Navigate to the AWS Lambda function in the AWS Console.

Step 2: Write the Code in the Lambda Function

Code to Stop an EC2 Instance: [language=Python] import boto3 re- gion = 'us-east-1' instances = ['i-12345cb6de4f78g9h']ec2= boto3.client('ec2', regionname = region) def lambdahandler(event,contex:ec2.stopinstances(InstanceIdsinstances)print(′Stoppedyourinstances :′ +str(instances))Code to Start an EC2 Instance: [language=Python] import boto3 region= 'us-west-1' instances = ['i-12345cb6de4f78g9h', 'i-08ce9b2d7eccf6d26'] ec2= boto3.client('ec2', regionname=region)def lambdahandler(event, context) : ec2.startinstances(InstanceIds=instances)print(′Startedyourinstances+str(instances))

Step 3: Deploy the Code

Deploy the Lambda function by uploading or saving the code.

Step 4: Test the Code

Run a test in the AWS Console or trigger the Lambda function to ensure the code works as expected.

Step 5: Verify Status

Confirm that the EC2 instances have started or stopped as intended. Successful execution confirms that EC2 automation through AWS Lambda is working.

5. BUILDING A SERVERLESS WEB APPLICATION

The building of serverless web application is implemented in the following steps [13]:

Step 1: Plan Your Application

Understand the requirements of your application and identify necessary components such as:

- User authentication
- Database
- File storage
- Business logic

Step 2: Set Up AWS Account and IAM Roles

Create an AWS account and set up Identity and Access Management (IAM) roles to define permissions for your services to interact with each other.

Step 3: Designing Your Serverless Backend with AWS Lambda

AWS Lambda will form the backbone of your serverless application. De- fine functions that Lambda will execute. Each function should have a single responsibility, adhering to microservices best practices.

Step 4: Setting Up API Gateway

API Gateway acts as the entry point for your application, routing HTTP requests to the appropriate Lambda functions. Define RESTful endpoints and connect them to their corresponding Lambda functions.

Step 5: Integrating DynamoDB

Design a schema for your database using DynamoDB, considering its NoSQL nature. Set up tables and integrate them with Lambda functions for data storage and retrieval.

Step 6: Adding Authentication

For user authentication, use Amazon Cognito, which integrates well with API Gateway and Lambda to provide secure access to API endpoints.

Step7: Setting Up Amazon S3 for File Storage

To store files such as images or documents, use Amazon S3. Configure S3 buckets and access permissions, and integrate S3 with Lambda functions for file operations.

Step 8: Deploying Your Application

Once your application components are configured and integrated, deploy your serverless application. AWS provides various deployment tools, including the AWS Serverless Application Model (SAM) and the AWS Management Con- sole.

Step 9: Monitoring and Maintenance

Utilize AWS Cloud Watch to monitor your application. Set up alarms and logs to track the health and performance of your application.

6. RESULTS AND DISCUSSION

The deployment of a serverless web application using AWS demonstrates the transformative capabilities of serverless computing in modern application development. This section outlines the results achieved and discusses their implications in terms of scalability, performance, cost-efficiency, and overall feasibility. The serverless architecture dynamically scaled based on traffic demands without manual intervention. The pay-as-you-go model resulted in significant cost savings compared to traditional server-based deployments. Automating the deployment process using AWS Cloud formation is reduced setup time by 60%. The Implementing IAM roles, API Gateway authorizations, and encrypted DynamoDB data storage are ensured secure operation. Real-world scenarios such as an e-commerce application is demonstrated seamless integration of services (e.g., API Gateway for front-end interaction and Lambda for back-end logic). The future improvements in serverless web application are cold start mitigation and advanced monitoring.

7. CONCLUSION

Deploying a serverless application using AWS Lambda, DynamoDB, and API Gateway through the AWS CLI simplifies infrastructure management while ensuring scalability and flexibility. By leveraging these services, developers can focus on writing application logic without worrying about provisioning or managing servers. The process involves setting up the necessary resources, writing and deploying Lambda functions, configuring API Gateway to expose the functions, and using DynamoDB for data storage. This approach provides a cost-effective, efficient, and scalable solution for building modern web applications.

REFERENCES

[1] https://docs.aws.amazon.com
[2] https://aws.amazon.com/serverless/sam
[3] https://aws.amazon.com/blogs
[4] https://stackoverflow.com.
[5] https://dev.to.
[6] https://aws.amazon.com/aws-cost-management
[7] https://aws.amazon.com/cloudformation
[8] https://aws.amazon.com/security
[9] https://aws.amazon.com/architecture/well-architected.
[10] https://aws.amazon.com/whitepapers.
[11] https://www.udemy.com, https://www.coursera.org,
[12] https://aws.amazon.com/security
[13] https://aws.amazon.com/training