Serverless Web Application using AWS Services

Prepared in the partial fulfillment of the Summer Internship Program on AWS

AT

Under the guidance of,

Mrs. Sumana,APSSDC

Submitted by

M. Lakshmi Devi

M. Sirisha

C. Seenu

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of my summer internship project at **Andhra Pradesh Skill Development Corporation (APSSDC)**. This opportunity has been an enriching and transformative experience for me, and I am truly thankful for the support, guidance, and encouragement I have received along the way.

First and foremost, I extend my sincere regards to **Mrs. Sumana**, my supervisors and mentors, for providing me with valuable insights, constant guidance, and unwavering support throughout the duration of the internship. Their expertise and encouragement have been instrumental in shaping the direction of this project.

I would like to thank the entire team at **Andhra Pradesh Skill Development Corporation (APSSDC)** for fostering a collaborative and innovative environment. The camaraderie, knowledge sharing, and feedback I received from my colleagues significantly contributed to the development and success of this project.

In conclusion, I am honored to have been a part of this internship program, and I look forward to leveraging the skills and knowledge gained to contribute positively to future endeavors.

Thank you.

Abstract

The "Serverless Web Application" project leverages the power of AWS's serverless architecture to create a highly scalable and cost-efficient web application. This project utilizes AWS Lambda for executing backend logic without the need for provisioning or managing servers, allowing developers to focus solely on writing code. By using Amazon API Gateway, the application exposes secure and reliable RESTful APIs that interact with Lambda functions, facilitating seamless communication between the frontend and backend components. This architecture ensures that the application can handle varying levels of traffic with ease, scaling automatically in response to user demand.

Amazon S3 serves as the storage backbone for the application, providing a secure, durable, and highly available repository for static assets such as HTML, CSS, JavaScript files, and user-generated content. S3's integration with other AWS services, such as CloudFront for content delivery and IAM for fine-grained access control, further enhances the performance and security of the web application. By storing static assets in S3, the project benefits from reduced latency and improved load times, resulting in a better user experience.

Overall, the "Serverless Web Application" project demonstrates the advantages of adopting a serverless architecture, including reduced operational complexity, automatic scaling, and cost savings. By combining AWS Lambda, Amazon API Gateway, and Amazon S3, the application achieves high availability and performance, making it a robust solution for modern web application development. This project showcases how leveraging AWS's serverless services can streamline development processes and deliver scalable, secure, and efficient web applications.

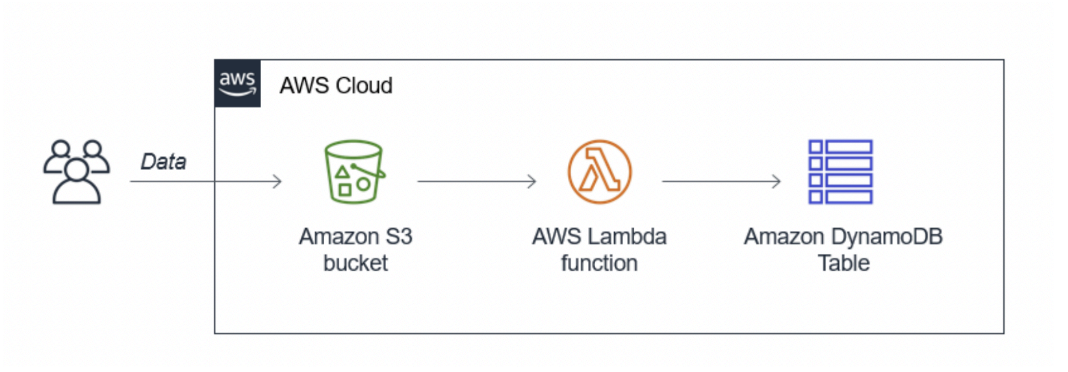
TABLE OF CONTENTS

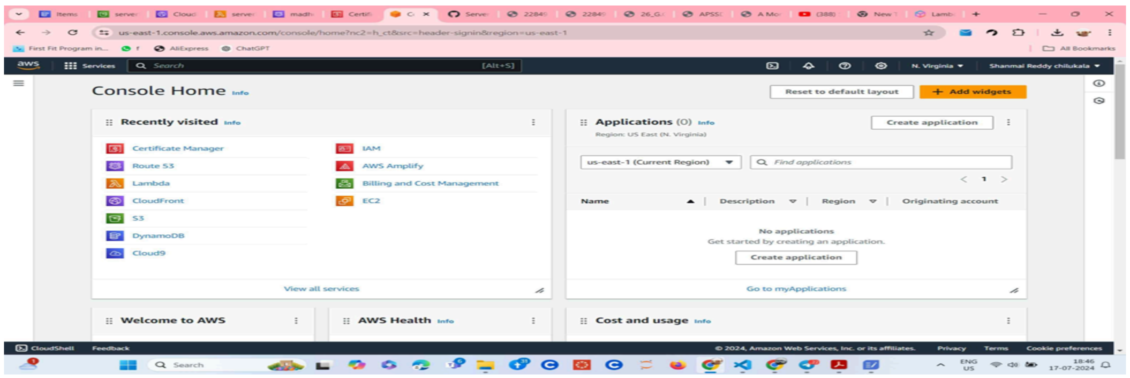
1.Introduction

In the contemporary digital era, web applications form the backbone of many business operations, providing platforms for interaction, data management, and service delivery. Traditional server-based architectures, however, often face challenges such as scalability, maintenance overhead, and cost efficiency. To address these issues, serverless architecture has emerged as a revolutionary approach, offering scalable and maintenance-free solutions. This project, titled "Serverless Web Application on AWS," demonstrates the creation of a robust, serverless web application that leverages the power of AWS services to deliver a seamless user experience while minimizing operational complexities.

The core objective of this project is to build a fully functional web application that allows users to perform Create, Read, Update, and Delete (CRUD) operations on a dataset stored in AWS DynamoDB. By employing AWS Lambda for backend processing, Amazon S3 for static file hosting, and CloudFront for content delivery, this project exemplifies how to integrate various AWS services into a cohesive, efficient, and scalable application.

AWS Services Used :





1. **AWS Lambda**:
   - Purpose: AWS Lambda is a serverless compute service that allows you to run code without provisioning or managing servers.
   - Functionality: It automatically scales your application by running code in response to each trigger, such as an HTTP request via API Gateway.
   - **Benefits**: Reduces the need for server maintenance, scales automatically with demand, and charges only for the compute time you consume.
2. **Amazon DynamoDB**:
   - Purpose: DynamoDB is a managed NoSQL database service that provides fast and predictable performance with seamless scalability.
   - Functionality: It is ideal for storing and querying large amounts of data with low latency and high throughput.
   - Benefits: Offers built-in high availability and fault tolerance, scales to handle high traffic, and supports flexible data models (key-value and document).
3. **Amazon S3**:
   - Purpose: Amazon Simple Storage Service (S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance.

   - Functionality: S3 is used to store and serve the web application's static files, such as HTML, CSS, and JavaScript.
   - Benefits: Provides durable storage, high availability, and scalable storage capacity. It also integrates with other AWS services and offers cost-effective storage options.
4. **Amazon CloudFront**:
   - Purpose: CloudFront is a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency and high transfer speeds.
   - Functionality: It caches the static files stored in S3 at edge locations around the world, ensuring faster delivery to users.
   - Benefits: Enhances the performance of the web application by reducing latency and improving load times. It also provides security features such as DDoS protection and encryption.

2. Methodology

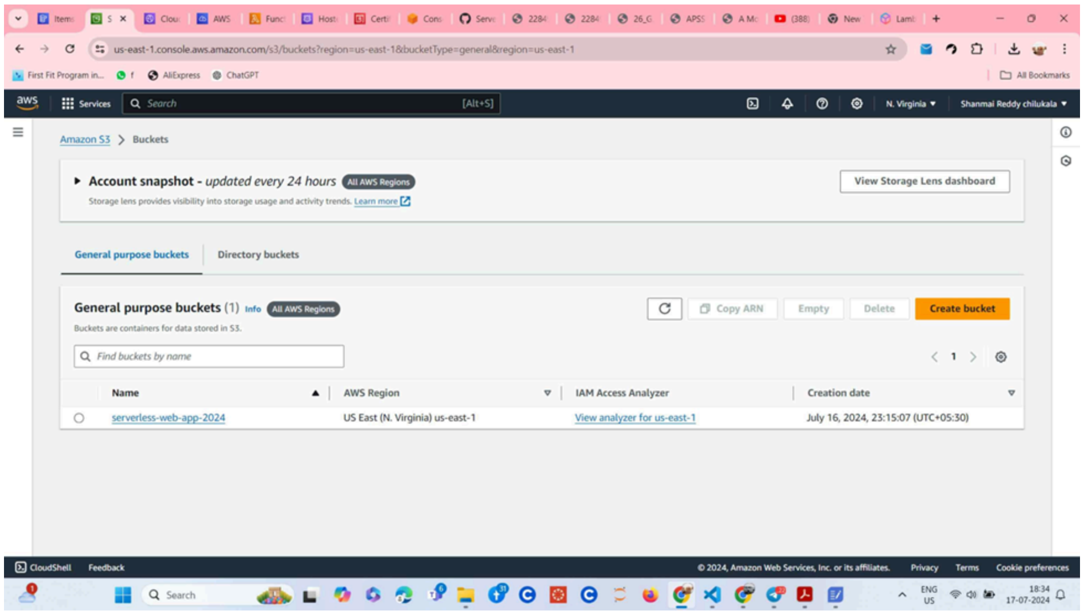
Steps to Build the Project

Building a serverless web application on AWS involves several well-defined steps. Here are the detailed steps to guide you through the process:

Step 1: Create an S3 Bucket in AWS Console and Upload Static Files

Objective: Set up a storage solution for your web application's static files (HTML, CSS, and JavaScript).

1. Log in to AWS Console: Open the AWS Management Console and log in with your credentials.
2. Navigate to S3: From the AWS services menu, search for and select "S3" to open the Amazon S3 console.
3. Create a New Bucket:
   - Click on the "Create bucket" button.
   - Enter a unique name for your bucket (e.g., `my-serverless-webapp-bucket`).
   - Choose a region where you want your bucket to be located.
   - Leave other settings as default for now and click on "Create bucket" at the bottom.
4. Upload Files to S3 Bucket:
   - Open your newly created bucket by clicking on its name.
   - Click on the "Upload" button.
   - In the upload dialog, click "Add files" and select all the HTML, CSS, and JavaScript files you need for your web application.
   - After selecting the files, click on "Upload" to start uploading the files to your S3 bucket.
5. Set Permissions for Static Website Hosting:
   - After uploading, go to the "Permissions" tab in your bucket settings.
   - Ensure that the bucket policy allows public read access to the files. Click "Edit" in the "Block public access" section and disable "Block all public access".
   - Save changes and confirm when prompted



us-east-1.console.aws.amazon.com/s3/buckets?region=us-east-1&bucketType=general&region=us-east-1

Amazon S3 > Buckets

Account snapshot - *updated every 24 hours* All AWS Regions

Storage lens provides visibility into storage usage and activity trends. Learn more

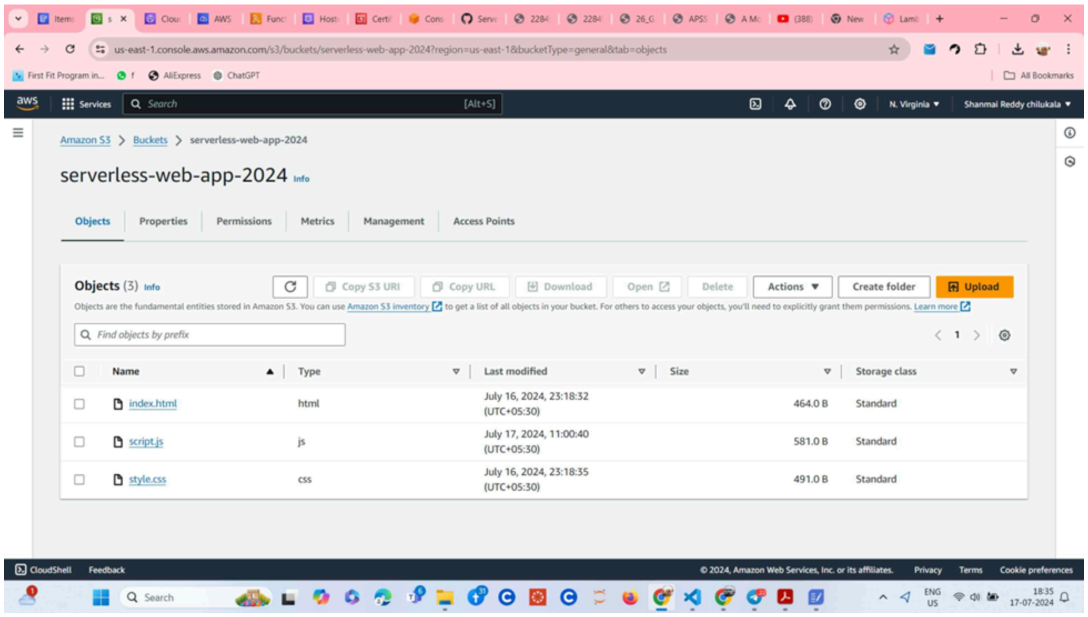
General purpose buckets | Directory buckets

General purpose buckets (1) Info All AWS Regions

Buckets are containers for data stored in S3.

Name	AWS Region	IAM Access Analyzer	Creation date
serverless-web-app-2024	US East (N. Virginia) us-east-1	View analyzer for us-east-1	July 16, 2024, 23:15:07 (UTC+05:30)

© 2024, Amazon Web Services, Inc. or its affiliates.



us-east-1.console.aws.amazon.com/s3/buckets/serverless-web-app-2024?region=us-east-1&bucketType=general&tab=objects

Amazon S3 > Buckets > serverless-web-app-2024

serverless-web-app-2024 Info

Objects | Properties | Permissions | Metrics | Management | Access Points

Objects (3) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
index.html	html	July 16, 2024, 23:18:32 (UTC+05:30)	464.0 B	Standard
script.js	js	July 17, 2024, 11:00:40 (UTC+05:30)	581.0 B	Standard
style.css	css	July 16, 2024, 23:18:35 (UTC+05:30)	491.0 B	Standard

© 2024, Amazon Web Services, Inc. or its affiliates.

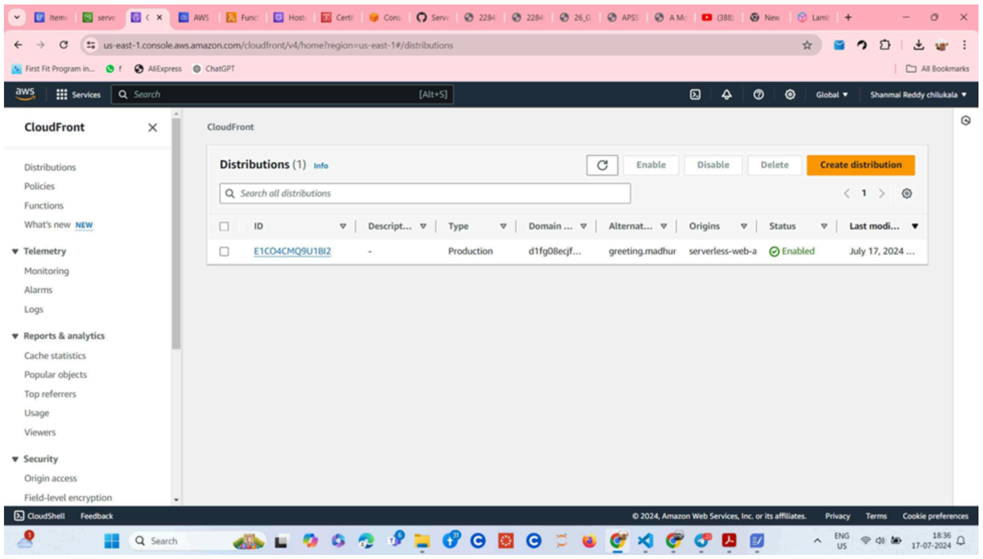
Step 2: Set Up the CloudFront Distribution and Integrate it with the S3 Bucket

Objective: Create a CloudFront distribution to serve the static files stored in the S3 bucket with low latency.

1. Log in to AWS Console: Open the AWS Management Console and navigate to "CloudFront."

2. Create a Distribution:
    - Click "Create Distribution" and choose "Web."
    - For "Origin Domain Name," select your S3 bucket.
3. Configure Settings:
    - Set "Viewer Protocol Policy" to "Redirect HTTP to HTTPS."
    - Adjust caching and other settings as needed.
4. Review and Create: Review your settings and click "Create Distribution."
5. Deploy: Wait for the status to change to "Deployed."
6. Update S3 Bucket Policy: Ensure the S3 bucket policy grants read permissions to CloudFront.

This step sets up CloudFront to globally deliver your S3-hosted static files with low latency.

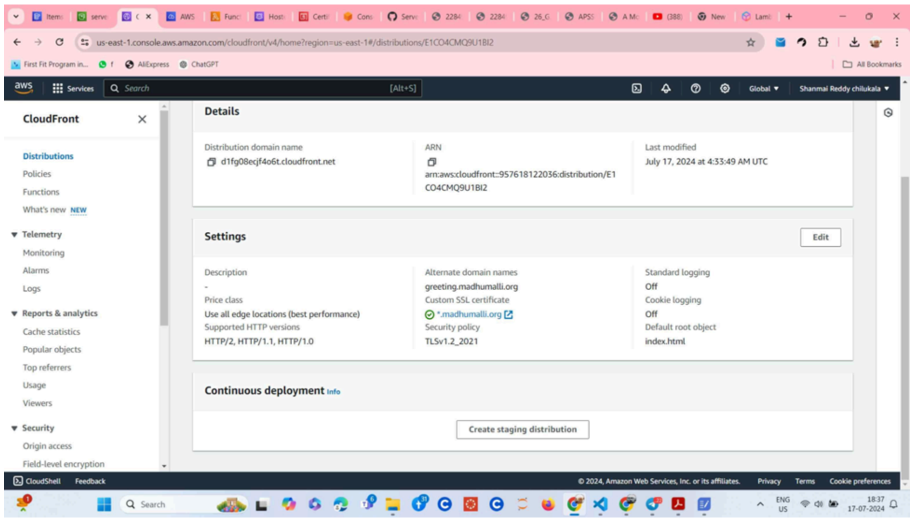


us-east-1.console.aws.amazon.com/cloudfront/v4/home?region=us-east-1#/distributions

Distributions (1) Info

ID	Descript...	Type	Domain ...	Alternat...	Origins	Status	Last modi...
E1CO4CMQ9U1BI2	-	Production	d1fg08ecjf...	greeting.madhur	serverless-web-a	Enabled	July 17, 2024 ...

© 2024, Amazon Web Services, Inc. or its affiliates.



us-east-1.console.aws.amazon.com/cloudfront/v4/home?region=us-east-1#/distributions/E1CO4CMQ9U1BI2

Details

Distribution domain name: d1fg08ecjf4o6t.cloudfront.net
ARN: arn:aws:cloudfront::957618122036:distribution/E1CO4CMQ9U1BI2
Last modified: July 17, 2024 at 4:33:49 AM UTC

Settings

Description: -
Price class: Use all edge locations (best performance)
Supported HTTP versions: HTTP/2, HTTP/1.1, HTTP/1.0
Alternate domain names: greeting.madhumalli.org
Custom SSL certificate: *.madhumalli.org
Security policy: TLSv1.2_2021
Standard logging: Off
Cookie logging: Off
Default root object: index.html

Continuous deployment Info

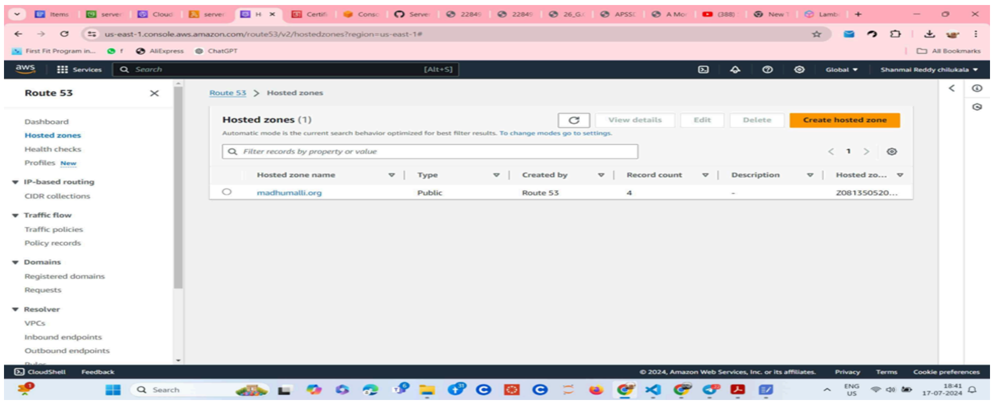
© 2024, Amazon Web Services, Inc. or its affiliates.

Step 3: Set Up AWS Route 53 and Certificate

Objective: Create a free domain, set up a hosted zone to manage DNS queries, and request and validate an SSL/TLS certificate.

1. **Log in to AWS Console**: Open the AWS Management Console and navigate to "Route 53."
2. **Register a Domain**:
   - Click on "Register Domain."
   - Search for a free domain name of your choice and complete the registration process.
3. **Create a Hosted Zone**:
   - After your domain registration is complete, go to the "Hosted zones" section.
   - Click "Create hosted zone" and enter your domain name. This tells Route 53 how to respond to DNS queries for your domain.
4. **Request a Certificate**:
   - Navigate to the AWS Certificate Manager (ACM) in the console.
   - Click "Request a certificate" and select "Request a public certificate."
   - Enter your domain name and complete the request process.
5. **Validate the Certificate**:
   - ACM will provide validation options. Choose DNS validation.
   - Add the provided CNAME record to your Route 53 hosted zone to validate your domain ownership.
6. **Integrate Route 53 with CloudFront**:
   - Go back to Route 53 and select your hosted zone.
   - Click "Create Record Set" and add an "A" record with an alias to your CloudFront distribution.

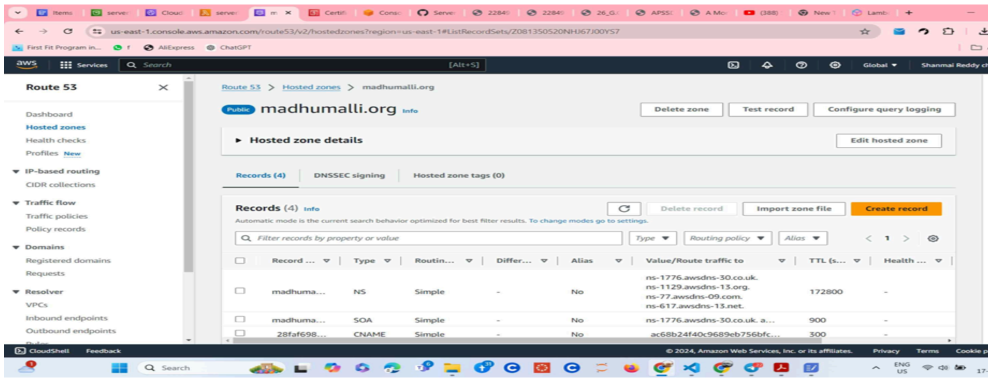
This step sets up Route 53 for domain management, requests and validates a certificate for secure connections, and integrates Route 53 with CloudFront to serve your application via your custom domain.



Hosted zones (1)

Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

Hosted zone name	Type	Created by	Record count	Description	Hosted zo...
madhumalli.org	Public	Route 53	4	-	Z081350520...



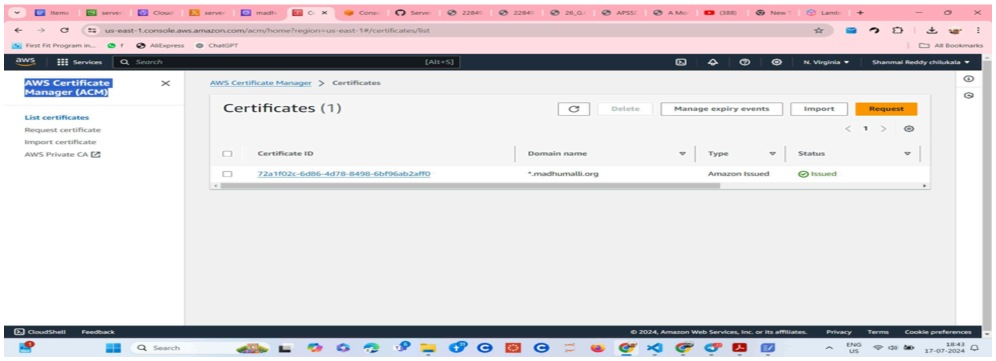
Route 53 > Hosted zones > madhumalli.org

Public madhumalli.org Info

Records (4)

Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

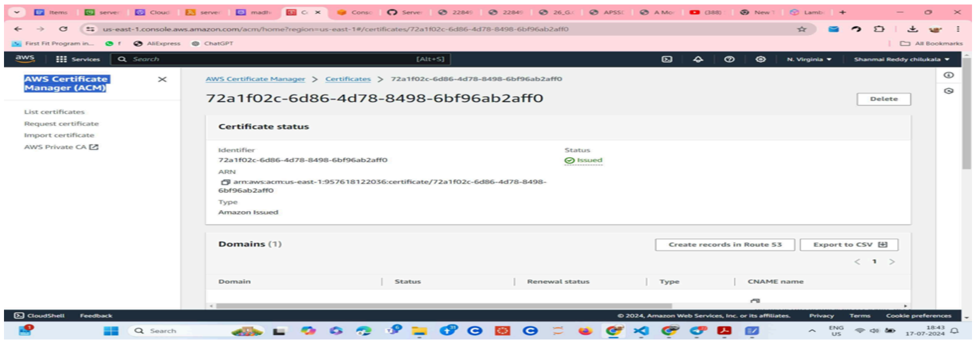
Record ...	Type	Routin...	Differ...	Alias	Value/Route traffic to	TTL (s...	Health ...
madhuma...	NS	Simple	-	No	ns-1776.awsdns-30.co.uk. ns-1129.awsdns-13.org. ns-77.awsdns-09.com. ns-617.awsdns-13.net.	172800	-
madhuma...	SOA	Simple	-	No	ns-1776.awsdns-30.co.uk. a...	900	-
28faf698...	CNAME	Simple	-	No	ac68b24f40c9689eb756bfc...	300	-



AWS Certificate Manager > Certificates

Certificates (1)

Certificate ID	Domain name	Type	Status
72a1f02c-6d86-4d78-8498-6bf96ab2aff0	*.madhumalli.org	Amazon Issued	Issued



AWS Certificate Manager > Certificates > 72a1f02c-6d86-4d78-8498-6bf96ab2aff0

72a1f02c-6d86-4d78-8498-6bf96ab2aff0

Certificate status

Identifier: 72a1f02c-6d86-4d78-8498-6bf96ab2aff0

Status: Issued

ARN: arn:aws:acm:us-east-1:957618122036:certificate/72a1f02c-6d86-4d78-8498-6bf96ab2aff0

Type: Amazon Issued

Domains (1)

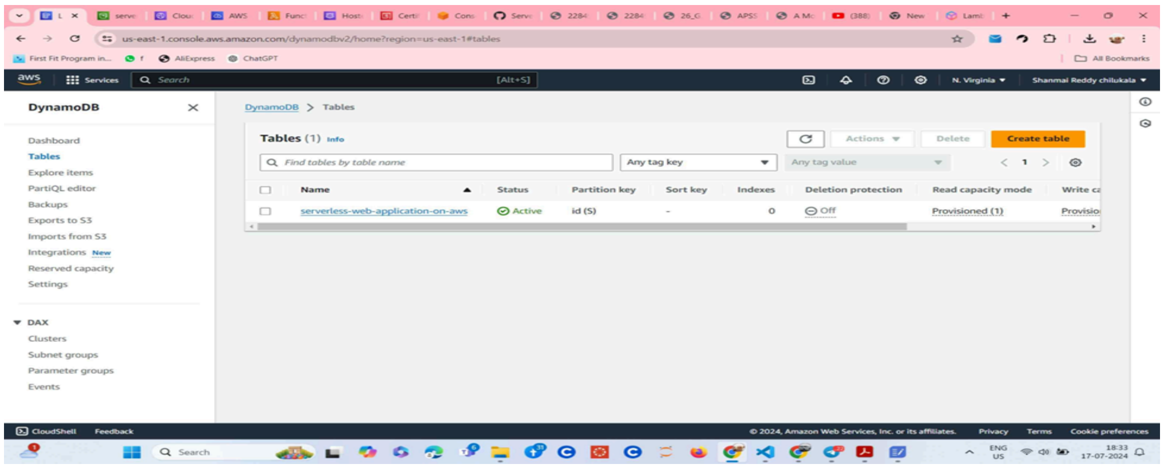
Domain	Status	Renewal status	Type	CNAME name

Step 4: Set Up AWS DynamoDB and Map with Lambda

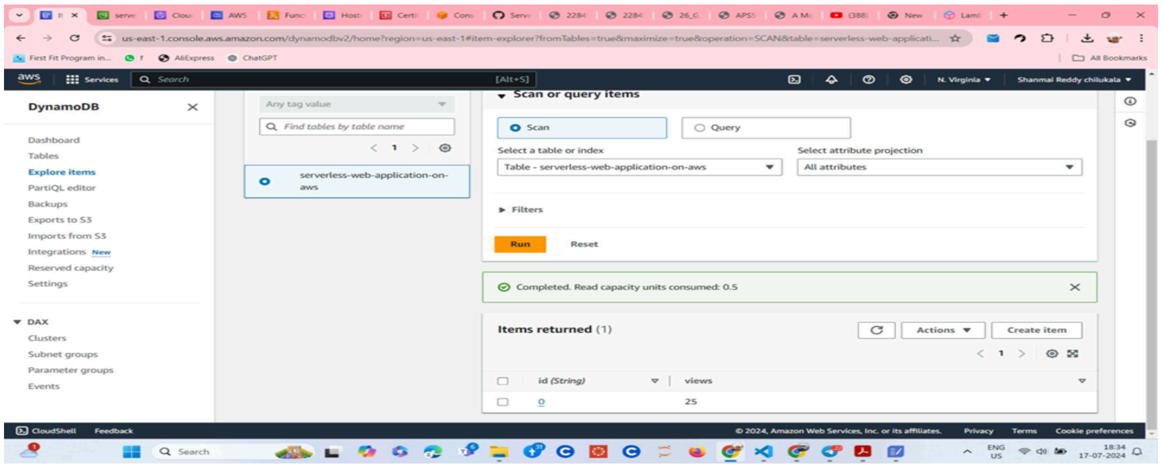
Objective: Create a DynamoDB table to store application data, configure AWS Lambda to perform CRUD operations on this table, and set up the necessary IAM roles.

1. **Log in to AWS Console**: Open the AWS Management Console and navigate to "DynamoDB."
2. **Create a DynamoDB Table**:
   - Click "Create Table."
   - Enter a table name and specify a primary key (e.g., `itemId`).
   - Configure any additional settings as required and click "Create."

3. **Create Lambda Functions**:
   - Navigate to "Lambda" in the AWS Console.
   - Click "Create function" and choose "Author from scratch."
   - Provide a name for your function and choose a runtime (e.g., Python, Node.js).
4. **Create an IAM Role for Lambda**:
   - Navigate to the IAM service in the AWS Console.
   - Click "Roles" and then "Create role."
   - Choose "Lambda" as the service that will use this role.
   - Attach the "AmazonDynamoDBFullAccess" policy to the role to allow access to DynamoDB.
   - Complete the role creation process and note down the role ARN.
5. **Attach the IAM Role to the Lambda Function**:
   - Go back to your Lambda function configuration.
   - In the "Execution role" section, select "Use an existing role" and choose the IAM role you created.
6. **Configure Lambda to Access DynamoDB**:
   - In the Lambda function, use the AWS SDK to perform CRUD operations on the DynamoDB table.



Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write ca
serverless-web-application-on-aws	Active	id (S)	-	0	Off	Provisioned (1)	Provisio



Scan [✓] Query

Select a table or index: Table - serverless-web-application-on-aws

Select attribute projection: All attributes

Filters

Completed. Read capacity units consumed: 0.5

Items returned (1)

id (String)	views
0	25

Step 5: Set Up AWS Lambda and Include the Code

Objective: Create and configure AWS Lambda functions to handle the backend logic of your web application.

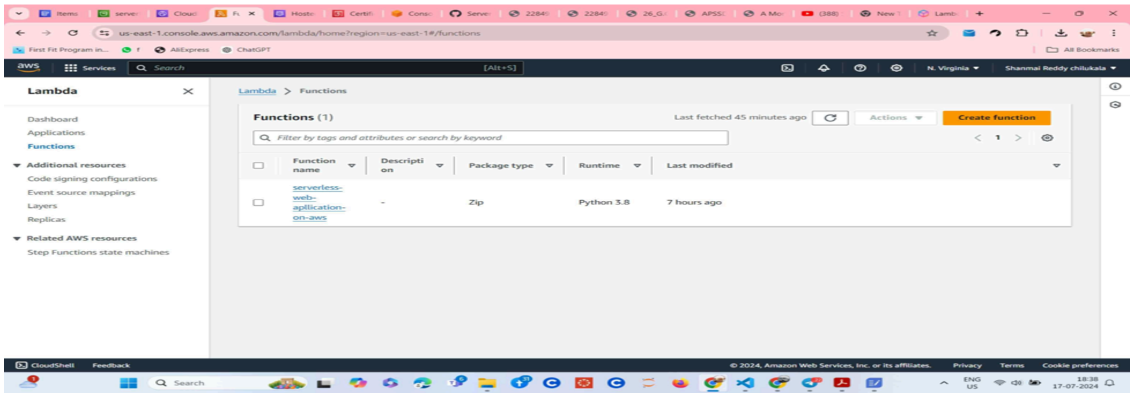
1. **Log in to AWS Console**: Open the AWS Management Console and navigate to "Lambda."
2. **Create a Lambda Function**:
   - Click "Create function."
   - Choose "Author from scratch."
   - Provide a function name and select a runtime (e.g., Node.js, Python).
   - Select "Create a new role with basic Lambda permissions" or choose an existing role with the necessary permissions.
3. **Configure the Lambda Function**:
   - In the function code editor, add your code to interact with DynamoDB.

4. **Test the Lambda Function**:

- Create a test event in the Lambda console to ensure your function works as expected.
- Click "Test" and review the output to verify the function's behavior.

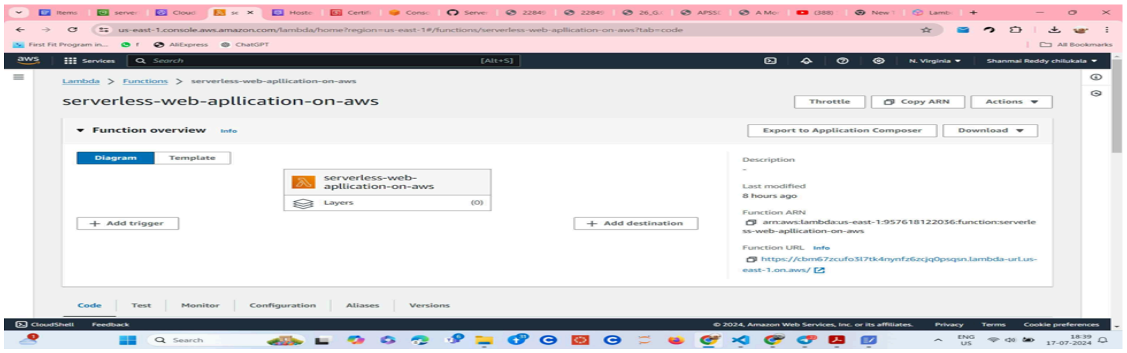
5. **Deploy the Function**:

- Once you have tested your function, click "Deploy" to save and activate the changes.



Functions (1)

Function name	Description	Package type	Runtime	Last modified
serverless-web-apllication-on-aws	-	Zip	Python 3.8	7 hours ago



serverless-web-apllication-on-aws

Description
-

Last modified
8 hours ago

Function ARN
arn:aws:lambda:us-east-1:957618122036:function:serverless-web-apllication-on-aws

Function URL
https://cbm67zcufo3l7tk4nynfz6zcjq0psqsn.lambda-url.us-east-1.on.aws/

Code :

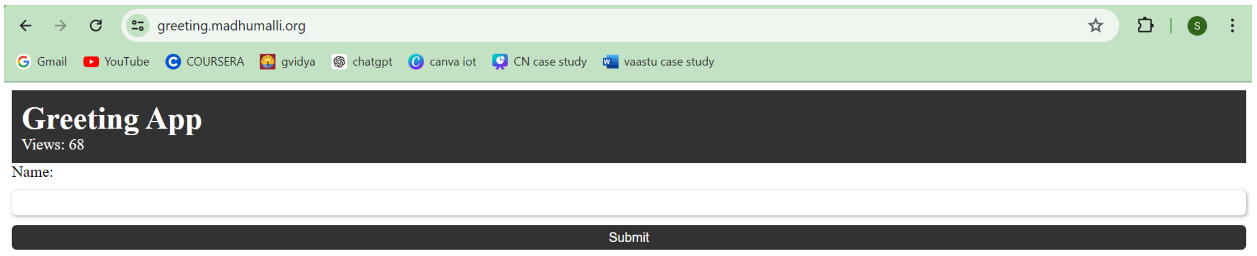
```
import json

import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('serverless-web-application-on-aws')

def lambda_handler(event, context):

    response = table.get_item(Key={

        'id':'0'

    })

    views = response['Item']['views']

    views = views +
print(views)

response = table.put_item(Item={

        'id':'0',

        'views': views    })

 return views
```

Step 6: Testing the Final Website and Integrating Lambda with Other Elements

Objective: Ensure the entire serverless web application is functional by testing the integration of all components, including AWS Lambda, DynamoDB, S3, CloudFront, and Route 53.

1. **Integrate Lambda with Other Services**:
    - **DynamoDB**: Ensure your Lambda functions can perform CRUD operations on the DynamoDB table.
    - **CloudFront**: Ensure CloudFront is correctly serving the static files from the S3 bucket and that the distribution is using the SSL/TLS certificate from ACM.
    - **Route 53**: Verify that your domain name in Route 53 points to the CloudFront distribution, and DNS queries are resolved correctly.
2. **Testing the Website**:
    - **Front-end Functionality**: Access your website via the domain name set in Route 53. Check that all static files (HTML, CSS, JavaScript) are loading correctly from the S3 bucket.
    - **API Endpoints**: Test each CRUD operation (Create, Read, Update, Delete) via the web interface to ensure Lambda functions are working correctly with DynamoDB.
    - **Security**: Verify that the website is served over HTTPS and the SSL/TLS certificate is correctly applied.
    - **Performance**: Check the load times and performance of your website through the CloudFront distribution.
3. **Debugging and Troubleshooting**:
    - **Logs**: Use AWS CloudWatch Logs to monitor Lambda function executions and identify any errors or issues.
    - **IAM Roles**: Ensure that all IAM roles and permissions are correctly set up to allow the necessary access between services.
    - **Configuration**: Double-check all configurations in S3, CloudFront, Route 53, DynamoDB, and Lambda to ensure they are correctly integrated.
4. **Final Deployment**:

- After successful testing, deploy the final version of your web application.
- Monitor the application for any issues and ensure it runs smoothly.

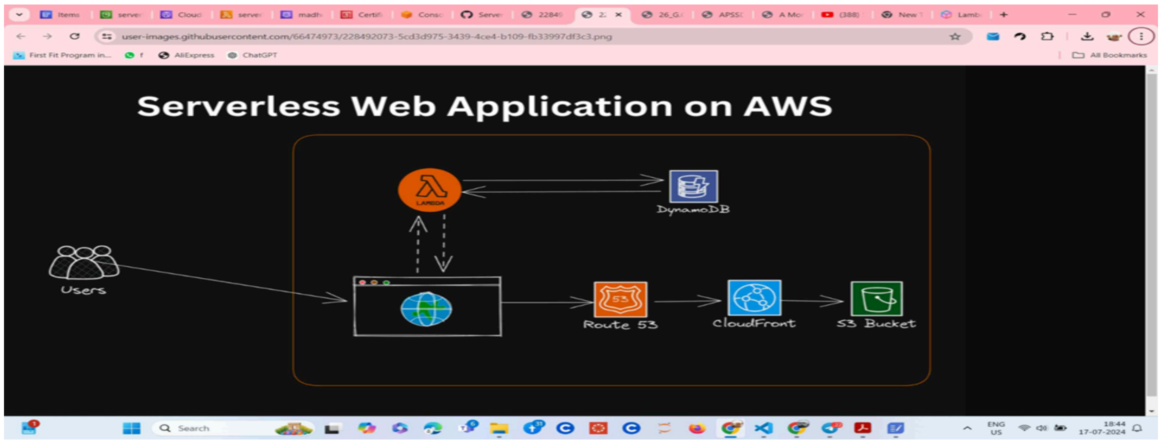


Greeting App

Views: 68

Name:

3.Architecture


user-images.githubusercontent.com/66474973/228492073-5cd3d975-3439-4ce4-b109-fb33997df3c3.png
Serverless Web Application on AWS
LAMBDA
DynamoDB
Users
Route 53
CloudFront
S3 Bucket

1. **AWS S3**:
   - Storage: All static files (HTML, CSS, JavaScript) are stored in an S3 bucket.
   - Hosting: S3 serves as the origin for the static content of the web application.
2. **AWS CloudFront**:
   - Content Delivery: CloudFront is configured to distribute the static files from the S3 bucket, ensuring low latency and high transfer speeds.
   - Custom Domain: CloudFront is associated with a custom domain provided by Route 53, making the site easily accessible.
3. **AWS Route 53**:
   - Domain Management: Route 53 manages the custom domain name for the web application.
   - DNS Services: It routes traffic to the CloudFront distribution, ensuring the site is secure and available.
4. **AWS Lambda**:
   - Serverless Compute: Lambda functions handle the backend logic, processing CRUD operations.
   - Integration: Lambda functions are integrated with DynamoDB to manage the application's data.
5. **AWS DynamoDB**:
   - Database: DynamoDB is used to store and retrieve application data.
   - Scalability: It provides a highly scalable and low-latency NoSQL database solution.

How it Works:

- File Storage and Delivery: Static files are uploaded to the S3 bucket. CloudFront is configured to cache these files and serve them globally, reducing latency.
- Custom Domain and Security: A custom domain is set up using Route 53, and an SSL/TLS certificate is requested and validated to ensure secure connections.
- Backend Processing: Lambda functions are created to perform CRUD operations on the DynamoDB table. These functions are triggered via API Gateway (if using HTTP) or directly from the web application.
- DNS Management: Route 53 handles DNS queries and directs traffic to the CloudFront distribution, ensuring the application is accessible via the custom domain.

Result

The serverless web application developed using AWS services demonstrates the efficiency and scalability of serverless architecture. By leveraging AWS S3 for storing static files, the application ensures high availability and reliability. CloudFront integrates seamlessly with S3, enabling fast content delivery across the globe with low latency. The custom domain managed by Route 53 provides a professional and accessible web presence, enhanced by secure connections via SSL/TLS certificates.

AWS Lambda functions, integrated with DynamoDB, efficiently handle backend processing, managing CRUD operations with ease. The use of IAM roles ensures secure access to DynamoDB tables, maintaining data integrity and security. Through comprehensive testing, the application showcases robust performance, effectively handling user interactions and data transactions.

This project not only achieves its primary goal of creating a serverless web application but also highlights the practical benefits of using AWS services. It provides a scalable, cost-effective solution that can handle varying loads without the need for manual server management. The application is now fully operational, delivering a seamless and secure user experience.

Conclusion

The serverless web application project underscores the advantages of adopting a serverless architecture using AWS services. By eliminating the need for traditional server management, the project demonstrates significant improvements in scalability, cost efficiency, and operational simplicity. AWS Lambda and DynamoDB streamline backend processes, while S3 and CloudFront ensure fast and reliable content delivery. The integration of Route 53 with a custom domain and SSL/TLS certificates enhances the application's security and accessibility.

Throughout the development and testing phases, the project highlighted the importance of proper configuration and integration of AWS services. The successful deployment of this application not only fulfills the project's objectives but also equips developers with practical insights into leveraging serverless technologies.

This experience reinforces the value of AWS's robust ecosystem, offering scalable and secure solutions for modern web applications. Future projects can build on this foundation, exploring further innovations in serverless computing and cloud-based architectures. The project thus serves as a testament to the potential of AWS services in delivering efficient and reliable serverless applications.